TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zranitelnost technologie Mifare Classic

Diplomová práce

Studijní program: N2612 – Elektrotechnika a informatika
Studijní obor: 1802T007 – Informační technologie

Autor práce: **Bc. Jan Petřvalský**
Vedoucí práce: doc. Ing. Milan Kolář, CSc.

Liberec 2014

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Mifare Classic Technology Vulnerabilities

Diploma thesis

Study programme: N2612 – Electrotechnology and informatics
Study branch: 1802T007 – Information technology

Author: **Bc. Jan Petřvalský**
Supervisor: doc. Ing. Milan Kolář, CSc.

Liberec 2014

Tento list nahraďte
originálem zadání.

Prohlášení

Byl jsem seznámen s tím, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu TUL.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Diplomovou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum:

Podpis:

Abstrakt

Technologie pro bezdrátovou identifikaci Mifare Classic má i přes známé bezpečnostní nedostatky stále významný tržní podíl. Na trhu se vyskytují starší transpondéry a také transpondéry od různých výrobců, které mohou mít různou úroveň zranitelnosti.

Tato práce se zaměřuje na analýzu známých nedostatků této technologie, identifikaci hlavních míst zranitelnosti a následným návrhem metodiky pro testování zranitelnosti transpondérů.

Získané poznatky byly využity při návrhu a implementaci nástroje pro testování zranitelnosti, který je postaven nad knihovnou Libnfc.

V poslední části byly navrženy změny sloužící k odstranění zranitelnosti. Prakticky by však bylo obtížné tyto změny snadno aplikovat, tak byla ještě navíc navržena metodika pro využití technologie Mifare Classic s důrazem na minimalizaci případných bezpečnostních rizik.

Klíčová slova:
Mifare Classic, Crypto1, kryptografie, bezpečnost, testování, Libnfc

Abstract

Technology for radio identification Mifare Classic has even through well known security issues still dominant position on market. It is possible to purchase old transponders and transponders from various manufacturers with different level of vulnerability.

This work focuses on analysis of known security issues of this technology, identifies main vulnerability spots and proposes testing methodics.

Acquired findings were used to design and implement vulnerability testing tool based on Libnfc library.

There were also proposed changes to remove vulnerability. It is practically very difficult to apply these changes, so there was also proposed methodics for using Mifare Classic with focus to minimize security risks.

Key words:
Mifare Classic, Crypto1, cryptography, security, testing, Libnfc

Poděkování

Děkuji Ing. Milanu Kolářovi, CSc. za cenné podněty, odborné konzultace a pomoc při úpravě formální stránky této práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zdrojových kódů

Seznam zkratek

RFID	Radio Frequency Identification	13
DOS	Denial of Service	13
REQA	Request Command, Type A	15
ATQA	Answer To Request	15
UID	Unique IDentifier	15
EEPROM	Electrically Erasable Programmable Read-Only Memory	16
NUID	Non-Unique IDentifier	17
CRC	Cyclic Redundancy Check	20
PRNG	Pseudorandom Number Generator	21
LFSR	Linear Feedback Shift Register	24
FPGA	Field Programmable Gate Array	29
SDK	Software Development Kit	33
I2C	Inter-Integrated Circuit	39
SPI	Serial Peripheral Interface	39
NFC	Near Field Communication	42
SAK	Select Acknowledge, Type A	50

1 Úvod do problematiky

Technologie RFID[1] se pomalu přesunuly z průmyslové sféry do každodenního života běžného člověka, aniž by si to uvědomoval. V současné době jsou různé standardy RFID přenosu využívány zejména k identifikaci osob (například při přístupu do budov), pro účely účtování přepravy a také drobné bezhotovostní platby.

Použití RFID karet je jednoduché a rychlé, jelikož uživatel nemusí znát žádné přístupové kódy či vkládat kartu do čtečky, jako je tomu u tradičních karet s magnetickými proužky. Čtení je bezkontaktní a stačí tedy, když uživatel přiloží kartu do blízkosti elektromagnetického pole čtečky.

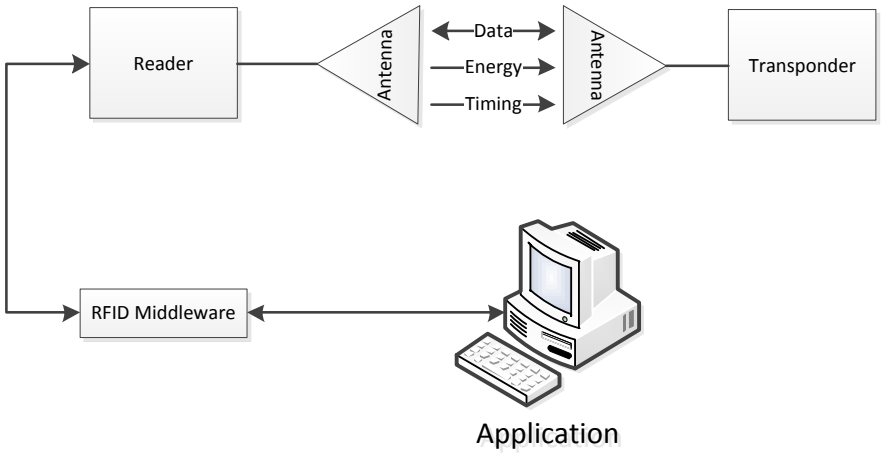
Z důvodu velice nízké ceny má i přes známé bezpečnostní nedostatky vysoký tržní podíl standard Mifare Classic.

Na obrázku 1.1 jsou zobrazeny prvky při typickém využití RFID. Uživatel prokazuje svojí identitu pomocí RFID karty, kterou přikládá ke čtečce. Čtečka z karty získá identifikátor uživatele a předá jej do další vrstvy aplikace. Následně je provedena autentizace, například ověřením pomocí dotazu do databáze s daným identifikátorem uživatele. V kladném případě je uživatel autorizován a jsou například otevřeny vstupní dveře, nebo vykonána jiná činnost.

Samozřejmě bezdrátová komunikace může být přijímána libovolným správně nastaveným zařízením v dosahu a do komunikace může být také zasahováno z nelegitimního zařízení. Obecně mohou být na bezdrátově komunikující zařízení prováděny následující útoky:

1. Odposlech
2. Opakování (replay attack)
3. Rušení komunikace, případně jiná forma DOS[2] útoku
4. Neoprávněné čtení a zápis

[1] Radio Frequency Identification
[2] Denial of Service



Obrázek 1.1: Typický případ použití RFID

K zamezení prostého odposlouchávání komunikace útočníkem je standard Mifare Classic vybaven proudovou šifrou Crypto-1. Veškerá komunikace mezi transpondérem a legitimní čtečkou je šifrovaná a útočník ji bez znalosti klíče nedokáže dešifrovat. Tato forma obrany však není účinná, pokud v dané aplikaci je uživatel rozpoznáván pouze na základě identifikátoru karty, který je přenášen v čistém textu.

Při provádění opakovacího útoku (replay attack) útočník přehrává komunikaci, kterou dříve získal odposlechem komunikace mezi transpondérem a legitimní čtečkou. Útočník dokonce ani nemusí být schopen získat obsah komunikace, poněvadž mu stačí znát pouze význam. Obrana je zde řešena pomocí autentizace bloku číslem z generátoru pseudonáhodných čísel. Šifrovaná komunikace pak vypadá pokaždé jinak[3].

Poněvadž by obrana proti záměrnému rušení komunikace byla obtížná, a tedy i nákladná, tak také nijak není řešena. Proti DOS útokům, jako je například použití velkého upravených množství karet, je možné se bránit na úrovni konkrétní aplikace.

Proti neoprávněnému čtení (identifikace osob) či změně (zvýšení zůstatku, snížení ceny zboží) je technologie Mifare Classic chráněna výše zmíněným šifrováním. Bez znalosti klíče by nemělo jít s pamětí jakkoliv manipulovat.

Na trhu se objevují různé implementace transpondérů tohoto standardu, které mají různou úroveň zranitelnosti. Tato práce se zabývá analýzou bezpečnostních nedostatků a návrhem nástroje, který by umožňoval v rámci možností technologie bezpečnost testovat.

[3]Záleží na rozsahu generátoru

2 Mifare Classic

Mifare Classic je aplikační protokol postavený nad transportní vrstvou realizovanou RFID standardem ISO 14443. Mifare Classic popisuje organizaci paměti transpondéru, komunikační protokol na aplikační vrstvě a šifrování přenosu.

2.1 Antikolizní protokol

Antikolizní protokol slouží k výběru jednoho z transpondérů, které jsou v dosahu vysílače.

Po vložení transpondéru do elektromagnetického pole vysílače transpondér čeká na výzvu k zahájení svého ohlášení čtečce. Díky tomuto přístupu se nemůže stát, že zařízení vysílala současně.

Čtečka v pravidelných intervalech odesílá 7bitový příkaz pro identifikaci REQA[1]. Po příjmu všechny transpondéry v dosahu najednou odešlou odpověď ATQA[2] a přestanou vysílat až do následujícího vyzvání.

Tím začíná antikolizní smyčka, jejíž smyslem je získat identifikátory UID[3] všech dostupných zařízení. Kroky antikolizní smyčky jsou následující:

1. Čtečka odešle příkaz SEL (Anti-collision CL1)
2. Všechny dostupné transpondéry odešlou své UID
3. Čtečka odešle příkaz SEL UID (Select CL1)
4. Vybraný transpondér odešle identifikátor standardu.

V případě, že bylo v dosahu více zařízení, tak mohlo v bodě 2 dojít ke kolizi. Kolize je detekována jako vysoká úroveň po celou dobu hodinového taktu. Čtečka zvolí hodnotu prvního bitu – prefix UID a odešle znovu požadavek na identifikaci,

[1] Request Command, Type A
[2] Answer To Request
[3] Unique IDentifier

navíc připojí prefix. Teď však odpovídají již jen zařízení s daným prefixem. Tento postup opakuje, dokud čtečka nezíská UID všech zařízení v dosahu.

Vytváření modulace signálu a problém s kolizemi řeší automaticky ovladač v jádru a použitý integrovaný obvod PN532.

Více detailů ohledně modulace, kódování a řešení kolizí lze nalézt v normě (ISO 14443-3) a dokumentaci k Mifare Classic od NXP Semiconductors (NXP Semiconductors, 2014).

Ve vytvořené aplikaci je možné využít antikolizní protokol řízený samotným ovladačem a knihovnou Libnfc, nebo využít čip PN532 pouze pro generování modulace a samotný antikolizní protokol provést ručně. To může být vhodné například, pokud je nutné průběh řídit manuálně nebo je potřeba mít detailní výpisy (při automatickém průběhu pomoci hardwaru není možné detaily získat).

Výstup při softwarovém řízení antikolizního protokolu je uveden v tabulce 2.1.

Tabulka 2.1: Průběh antikolizního protokolu

Akce	Hex	Význam
Sent bits:	26 (7 bits)	Request type A
Received bits:	04 00	Response type A
Sent bytes:	93 20	Select All
Received bytes:	7d e7 c1 78 23	UID
Sent bytes:	93 70 7d e7 c1 78 23 6b 2e	Select UID
Received bytes:	08 b6 dd	Mifare Classic 1k

2.2 Organizace paměti

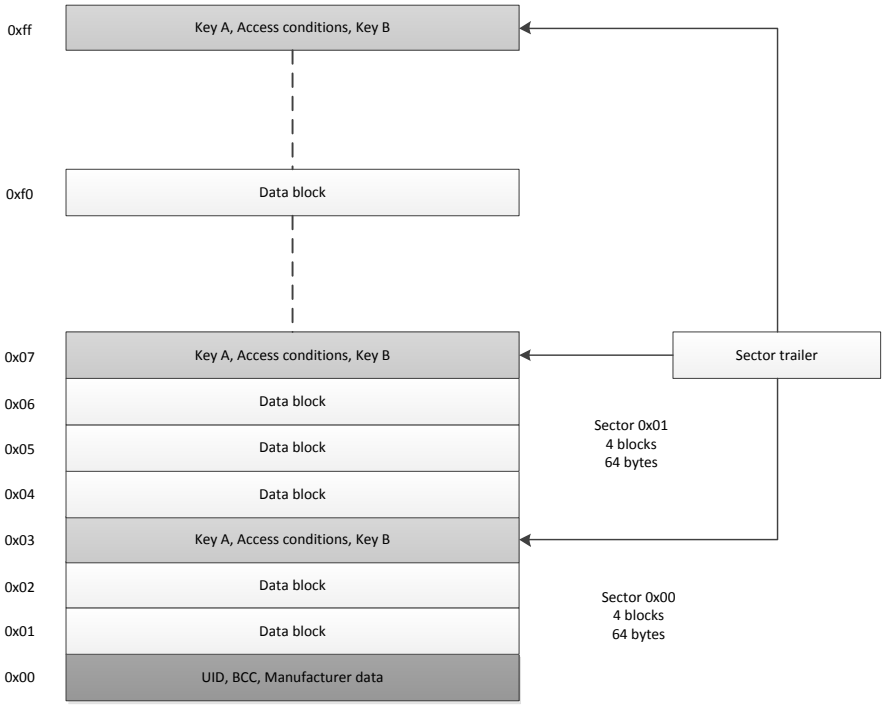
Datová paměť je u karet technologie Mifare Classic realizována pamětí typu EEPROM[4] o velikosti 1 kB. Paměť (Obrázek 2.1) je rozdělena do šestnácti čtyřblokových sektorů. Každý sektor se skládá ze čtyř bloků, do kterých lze uložit až 16 bajtů aplikačních dat.

První sektor prvního bloku je rezervován výrobcem pro uložení provozních informací.

2.2.1 Blok výrobce

V sektoru 0 se v bloku 0 nachází první datový blok. Tento blok je rezervován výrobcem pro uložení informací o kartě, z nichž nejdůležitější informací je číslo

[4]Electrically Erasable Programmable Read-Only Memory



Obrázek 2.1: Organizace paměti

NUID[5] uložené na prvních čtyřech bajtech.

Tento identifikátor je použit pro rozlišení jednotlivých zařízení při provádění antikolizního protokolu. Podle dokumentace (NXP Semiconductors, 2011) nemusí být tento identifikátor nutně unikátní.

Výrobce při produkčním testu tento blok naprogramoval a již není možné jeho obsah měnit nehledě na nastavení přístupových bitů (NXP Semiconductors, 2011).

2.2.2 Datový blok

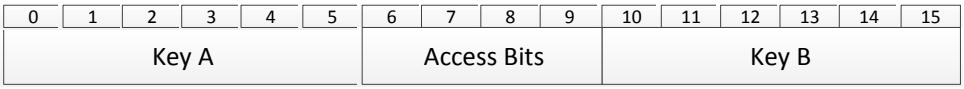
Pro uložení aplikačních dat je možné využívat datové bloky. V každém boku je k dispozici 16 bajtů, které jsou ve výchozí konfiguraci využitelné všechny.

Datový blok může být nakonfigurován do čtyřbajtového režimu nazývaného „value block“, pro tuto práci však není tato vlastnost příliš důležitá. Více informací lze nalézt v (NXP Semiconductors, 2011, s. 9).

[5]Non-Unique IDentifier

2.2.3 Konfigurační blok

Každý ze šestnácti sektorů obsahuje jeden konfigurační blok nazývaný sector trailer. V tomto bloku jsou uloženy šifrovací klíče (Key A a Key B) a přístupová práva k jednotlivým blokům daného sektoru. Konfigurační blok je zobrazen na obrázku 2.2.



Obrázek 2.2: Konfigurační blok

Pokud aplikace nevyužívá klíč B, tak je možné na příslušné paměťové místo po správném nastavení přístupových práv uložit aplikační data. Data mohou být uložena také na devátém bajtu a podle (NXP Semiconductors, 2011) pro ně platí stejná oprávnění jako pro bajty číslo 6, 7 a 8. Je nutné, aby byl vždy dodržen formát konfiguračních bitů pro nastavení přístupových práv.

2.3 Ochrana paměti

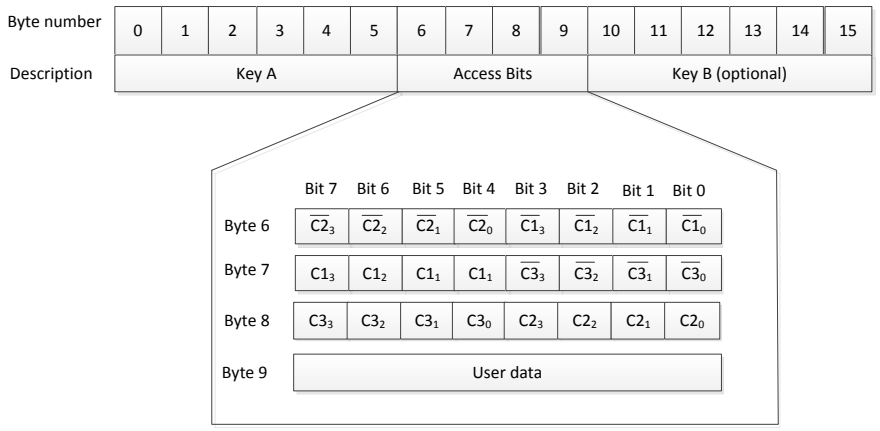
Šifrovací klíče A a B slouží pouze k šifrování přenosu, samotná data jsou v paměti uložena v podobě čistého textu. Standard Mifare Classic umožňuje pomocí tří bitů nastavit následující oprávnění:

- Čtení klíčem X
- Zápis klíčem X
- Čtení oběma klíči
- Zápis oběma klíči

kde X je A, nebo B.

Na obrázku 2.3 jsou znázorněny jednotlivé bity konfiguračního registru. Pro každý bit $C_{i,j}$ platí, že i je pozice konfiguračního bitu a j značí příslušnost k bloku.

Jednotlivé bity jsou zde uloženy v normální podobě jako $C_{i,j}$ a v invertované podobě $\overline{C}_{i,j}$. Podle citeMF1S503x, 11 při každém přístupu do paměti interní logika ověřuje strukturu. Pokud detekuje porušení formátu, tak nevratně zablokuje celý sektor.



Obrázek 2.3: Konfigurace přístupu

2.4 Příkazy aplikačního protokolu

Příkazy (uvedené v tabulce 2.2) aplikačního protokolu byly získány pomocí reverzního inženýrství a publikovány v práci (Nohl et al., 2008). V současné době je možné získat detalní popis všech příkazů i s použitým časováním z oficiální dokumentace (NXP Semiconductors, 2011).

Tabulka 2.2: Příkazy Mifare Classic

Příkaz	Hex	Popis
AUTH A	60 xx CRC	Autentizace bloku xx klíčem B
AUTH B	61 xx CRC	Autentizace bloku xx klíčem B
READ	30 xx CRC	Čtení bloku xx
WRITE	A0 xx CRC	Zápis bloku dat do bloku xx
HALT	50 00 CRC	Vypnutí transpondéru
ACK	A	Potvrzeno
NACK	4	Zamítnuto
NACK	5	Zamítnuto (chyba přenosu)

2.5 Zabezpečení přenosu

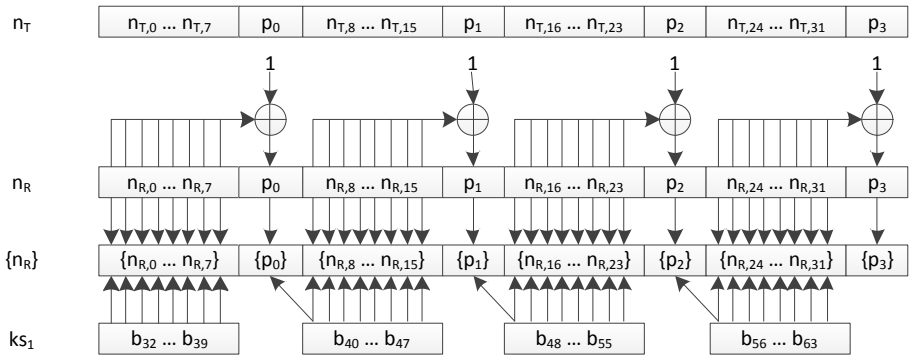
Norma ISO 14443A předepisuje, že za každým datovým bajtem musí následovat jeden bit liché parity. Pokud komunikace již probíhá šifrovaně, tak je paritní bit zašifrován stejným bitem proudu klíče jako následující datový bit. V tomto formátu

jsou data přenášena nejen při běžné komunikaci, ale také při provádění autentizačnímho protokolu.

Na obrázku 2.4 je schématicky zobrazen výpočet hodnoty paritního bitu a jeho šifrování v druhém kroku autentizačního protokolu.

Na aplikační vrstvě je pak další zabezpečení správnosti přenosu zajištěno pomocí 16bitového CRC[6], které však nemusí být nutně přidáváno za každý blok dat.

Programátor koncové aplikace nemusí tato zabezpečení přenosu manuálně řešit v programu, pokud to není vyžadováno, poněvadž o generování a kontrolu správnosti paritních bitů se stará sám integrovaný obvod ve čtečce. Kontrolní součet CRC pak může být taktéž generován čipem.



Obrázek 2.4: Výpočet parity a její šifrování

[6]Cyclic Redundancy Check

3 Crypto1

3.1 Generátor pseudonáhodných čísel

Při autentizaci bloků Mifare Classic používá protokol založený na challenge-response. Každá strana tedy musí mít možnost vygenerovat 32bitovou autentizační hodnotu nonce označovanou jako n_T resp. n_R.

Z bezpečnostních důvodů je nutné, aby tato hodnota byla náhodná, proto na obou komunikujících zařízeních musí být přítomen mechanismus ke generování náhodných resp. psudonáhodných čísel . Generátor použitý v zařízeních standardu Mifare Classic může být popsán pomocí generujícího polynomu $G(x) = x^{16} + x^{14} + x^{13} + x^{11} + 1$.

Definice 1 *Funkce zpětné vazby generátoru* $L : F_2^{32} \to F_2$ *je definována jako* $L(x_0, ..., x_{31}) = x_{16} \oplus x_{18} \oplus x_{19} \oplus x_{21}$

Definice 2 *Funkce* $F_2^{32} \to F_2^{32}$ *pro výpočet odpovědi při autentizaci je definována jako*
$suc(x_0, ..., x_{31}) = (x_1, ..., x_{31}, L(x_0, ..., x_{31}))$
$suc^n(x_0, ..., x_{31}) = suc^{n-1}(suc(x_0, ..., x_{31}))$

Definice 3 *Hodnota je platnou hodnotou pro autentizaci tehdy a jen tehdy, když* $\forall k \ \in \{0, 1, ..., 15\} : n_k \oplus n_{k+2} \oplus n_{k+3} \oplus n_{k+5} \oplus n_{k+16} = 0$

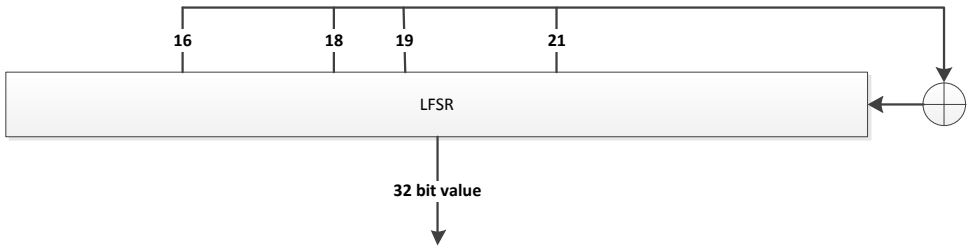
V samotném čipu je tento PRNG[1] implementován pomocí posuvného registru (Obrázek 3.1) s lineární zpětnou vazbou (Definice 1).

V rámci autentizačního protokolu jsou počítány odpovědi podle definice 2.

Podle (Garcia et al., 2008) závisí při komunikaci okamžik generování nové hodnoty na straně, kde je generátor použit. V případě čtečky je nová hodnota generována až při požadavku na nové náhodné číslo, oproti tomu transpondér vytváří novou hodnotu s každou náběžnou hranou hodinového signálu.

Dále je důležité zmínit, že po restartu zařízení generátor vytváří pokaždé stejnou sekvenci pseudonáhodných hodnot.

[1] Pseudorandom Number Generator



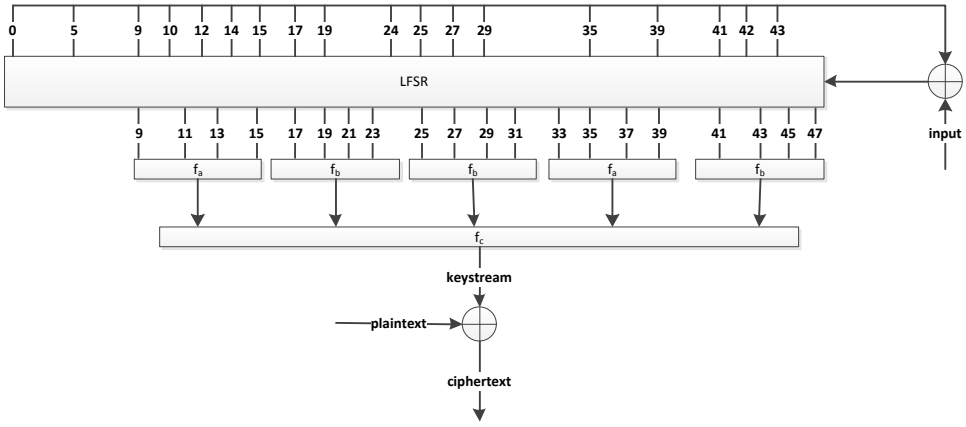
Obrázek 3.1: Generátor pseudonáhodných čísel

3.2 Proudová šifra

Pro šifrování komunikace v Mifare Classic je použita proprietární šifra Crypto1 (Obrázek 3.2).

Oficiální implementace této šifry nebyla nikdy výrobcem zveřejněna a je k dispozici pouze popis získaný pomocí reverzního inženýrství v pracích (Garcia et al., 2008) a (Nohl et al., 2008).

Šifra Crypto1 je velice jednoduchá proudová šifra tvořená registrem s lineární zpětnou vazbou a výstupními filtrovými funkcemi.



Obrázek 3.2: Šifra Crypto1

Pro odvození následujícího stavu posuvného registru je použit generující polynom $G(x) = x^{48} + x^{43} + x^{39} + x^{38} + x^{36} + x^{34} + x^{33} + x^{31} + x^{29} + x^{24} + x^{23} + x^{21} + x^{19} + x^{13} + x^9 + x^7 + x^6 + x^5 + 1$.

Definice 4 *Funkce zpětné vazby* $L : F_2^{48} \to F_2$ *je definována jako*
$L(x_0, x_1, x_{47}) = x_0 \oplus x_5 \oplus x_9 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{15} \oplus x_{17} \oplus x_{19} \oplus x_{24} \oplus x_{25} \oplus x_{27} \oplus x_{29} \oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{42} \oplus x_{43}$

Definice 5 *Výstupní nelineární funkce* $f : F_2^{48} \to F_2$ *je definována předpisem*
$f(x_0, x_1, ..., x_{47}) = f_c($
$f_a(x_9, x_{11}, x_{13}, x_{15}),$
$f_b(x_{17}, x_{19}, x_{21}, x_{23}),$
$f_b(x_{25}, x_{27}, x_{29}, x_{31}),$
$f_a(x_{33}, x_{35}, x_{37}, x_{39}),$
$f_b(x_{41}, x_{43}, x_{45}, x_{47}))$

Definice 6 *Filtrové funkce* f_a, f_b *a* f_c *jsou definovány následnovně*
$f_a(y_0, y_1, y_2, y_3) = ((y_0 \vee y_1) \oplus (y_0 \wedge y_3)) \oplus (y_2 \wedge ((y_0 \oplus y_1) \vee y_3))$
$f_b(y_0, y_1, y_2, y_3) = ((y_0 \wedge y_1) \vee y_2) \oplus ((y_0 \oplus y_1) \wedge (y_2 \vee y_3)$
$f_c(y_0, y_1, y_2, y_3, y_4) = (y_0 \vee ((y_1 \vee y_4) \wedge (y_3 \oplus y_4))) \oplus ((y_0 \oplus (y_1 \wedge y_3)) \wedge ((y_2 \oplus y_3) \vee (y_1 \wedge y_4)))$

Tvrzení 1 *Všechny filtrové funkce generují hodnotu logická 1 s pravděpodobností* $\frac{1}{2}$.

Důkaz 1 *Nechť* y_0, y_1, y_2 *a* $y_3 \in F_2$ *jsou jsou náhodné s rovnoměrným rozdělením a navzájem nezávislé proměnné. Pak pravděpodobnost* $Pr[f_a(y_0, y_1, y_2, y_3) = 1] =$
$= Pr[(((y_0 \vee y_1) \oplus (y_0 \wedge y_3)) \oplus (y_2 \wedge ((y_0 \oplus y_1) \vee y_3))) = 1] =$
$= Pr[(y_0 \vee y_1) \oplus (y_0 \wedge y_3) \neq (y_2 \wedge ((y_0 \oplus y_1) \vee y_3))]$

$y_0 = 0, y_1 = 0 :$
$Pr[0 \neq y_2 \wedge y_3] = Pr[y_2 \wedge y_3 = 1] = Pr[y_2 = 1] \wedge Pr[y_3 = 1] = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$

$y_0 = 0, y_1 = 0 :$
$Pr[1 \oplus 0 \neq (y_2 \wedge (1 \vee y_3))] = Pr[1 \neq y_2] = Pr[y_2 = 0] = \frac{1}{2}$

$y_0 = 1, y_0 = 0 :$
$Pr[1 \oplus y_3 \neq y_2 \wedge 1] = Pr[y_3 \oplus 1 \neq y_2] = Pr[\neg y_3 \neq y_2] = Pr[y_3 =_2] = Pr[y_3 = 0 | y_2 = 0] + Pr[y_3 = 1 | y_2 = 1] = \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}$

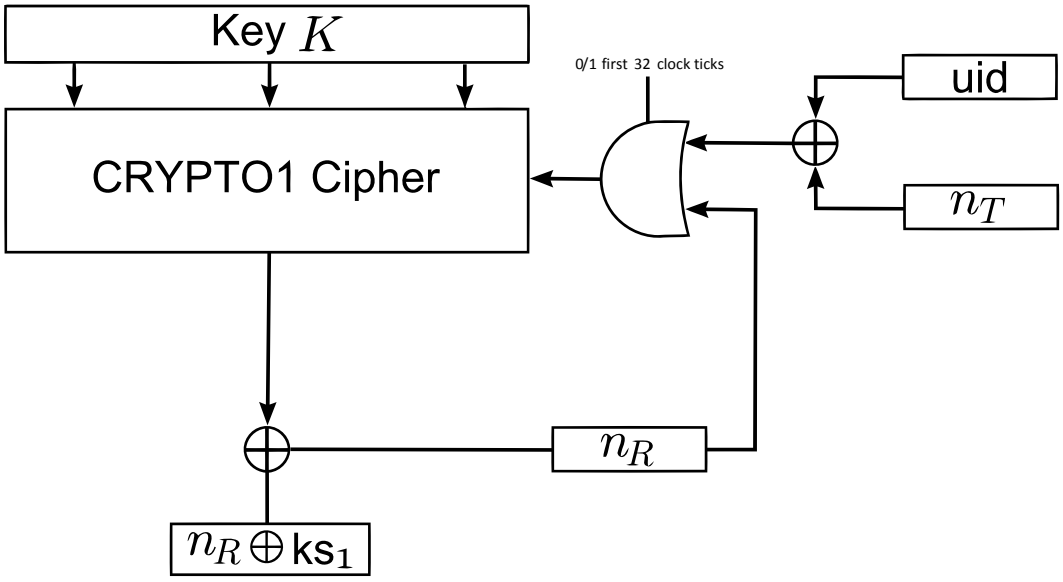
$y_0 = 1, y_0 = 1 :$
$Pr[1 \oplus y_3 \neq y_2 \wedge (0 \vee y_3)] = Pr[y_3 \oplus 1 \neq y_2 \wedge y_3] = Pr[y_3 = y_2 \wedge y_3] = 1 - Pr[y_2 = 0] \cdot Pr[y_3 = 1] = 1 - \frac{1}{2} \cdot \frac{1}{2} = 1 - \frac{1}{4} = \frac{3}{4}$

A celkem tedy získáváme $Pr[f_a(y_0, y_1, y_2, y_3) = 1] = \frac{1}{4} \cdot \frac{1}{4} + \frac{1}{4} \cdot \frac{1}{2} + \frac{1}{4} \cdot \frac{1}{2} + \frac{1}{4} \cdot \frac{3}{4} = \frac{1}{16} + \frac{1}{8} + \frac{1}{8} + \frac{3}{16} = \frac{1}{2}$. *Obdobně lze tvrzení dokázat pro ostatní filtrové funkce.*

Při každé hraně hodinového signálu[2] je celý registr posunut doleva a na pozici nejméně významného bitu r_{47} je zapsána hodnota ze zpětnovazební funkce $L(x)$. Pokud v čase k je stav LFSR[3] $r_k, r_{k+1}, ..., r_{k+47}$ a vstupní bit je i, pak v čase $k+1$ je stav registru $r_{k+1}, r_{k+2}, r_{k+3}, ..., r_{k+48}$.

Hodnota bitu nejvíce vpravo je $r_{k+48} = x_r \oplus x_{r+5} \oplus x_{r+9} \oplus x_{r+10} \oplus x_{r+12} \oplus x_{r+14} \oplus x_{r+15} \oplus x_{r+17} \oplus x_{r+19} \oplus x_{r+24} \oplus x_{r+25} \oplus x_{r+27} \oplus x_{r+29} \oplus x_{r+35} \oplus x_{r+39} \oplus x_{r+41} \oplus x_{r+42} \oplus x_{r+43} \oplus i$.

Vstupní bit i je využit pouze při inicializaci (Obrázek 3.3). Při běžném provozu se registr vyvíjí jen na základě aktuálního stavu.



Obrázek 3.3: Inicializace šifry Crypto1

Aktuální hodnota proudu klíče je vypočtena ze stavu posuvného registru. Čtveřice vybraných bitů jsou předány do filtrových funkcí f_a a f_b. Z nich je získáno pět hodnot, které jsou předány výstupní filtrové funkci f_c vracející jeden bit proudu klíče.

Pro šifrování či dešifrování je proveden exkluzivní součet proudu klíče s čistým textem.

[2]V případě čtečky při požadavku na novou hodnotu nikoliv hraně hodinového signálu.
[3]Linear Feedback Shift Register

3.3 Autentizační protokol

Po vybrání transpondéru během antikolizního protokolu je možné dále pokračovat v komunikaci pouze dvěma způsoby. První možností je ukončení komunikace odesláním příkazu HALT a druhou možností je provést autentizační protokol.

Pomocí tohoto protokolu, který je typu výzva-odpoveď (challenge-response), obě strany prokazují znalost sdíleného symetrického klíče. Pokud jedna ze stran klíč nezná, tak protokol není úspěšně dokončen a nelze číst data.

Při autentizačním protokolu jsou provedeny následující kroky:

1. Čtečka zasílá požadavek na autentizaci bloku klíčem A, nebo B.

2. Transpondér na základě konfiguračních bitů požadavek zamítne a odešle chybovou hlášku v čistém textu. Pokud je požadavek platný, tak přečte z PRNG náhodné číslo n_T a odešle ho protistraně.

   Obě strany nyní mají k dispozici symetrický klíč K, identifikátor UID a hodnotu výzvy n_T. Do registru šifry Crypto1 je přímo načten klíč K a následně je postupně pomocí zpětné vazby načtena hodnota $uid \oplus n_T$.

   Další komunikace od této chvíle probíhá šifrovaně.

3. Čtečka vygeneruje svojí vlastní výzvu n_R a nahraje ji postupně pomocí zpětné vazby do registru. Je důležité poznamenat, že hodnota n_R tedy přímo ovlivňuje generovaný proud klíče a při šifrování n_R. Tedy bity více vlevo ovlivňují bity více vpravo.

   Čtečka vypočte odpověď a_R na výzvu transpondéru n_T a odešle ji společně s výzvou n_R.

4. Transpondér vypočte a_T na výzvu n_R a odešle ji šifrovaně čtečce.

5. Stav šifry je na obou stranách stejný a je tedy možné datový přenos šifrovat. Od této chvíle se stav šifry vyvíjí již pouze na základě zpětné vazby.

Reálný průběh autentizace je znázorněn v tabulce 3.1.

Tabulka 3.1: Průběh autentizačního protokolu

Akce	Hex	Význam
Sent bytes:	60 30 76 4a	auth(block 30)
Received bytes:	42 97 c0 a4	n_T
Sent bytes:	7d db 9b 83 67 eb 5d 83	$n_R \oplus ks_1$, $a_R \oplus ks_2$
Received bytes:	8b d4 10 08	$a_T \oplus ks_3$

3.4 Zranitelnosti

Nedostatky v návrhu technologie Mifare Classic vedly k objevení mnoha zranitelností a jejich využití v konkrétních útocích. První útoky vyžadovaly aktivní sběr dat při komunikaci mezi legitimní čtečkou a transpondérem (Koning Gans et al., 2008), avšak později byly objeveny i útoky, které odposlouhávání komunikace již nevyžadují (Garcia et al., 2009) a (Courtois, 2009).

Slabiny pak lze rozdělit do skupin podle místa výskytu:

1. Slabiny v generátoru pseudonáhodných čísel – umožňují manipulaci s generovanou hodnotou, která pak již není náhodná
2. Slabiny samotné šifrovací funkce – slabiny v samotné proudové šifře a její výstupní funkci
3. Slabiny v komunikačním protokolu – zranitelnosti vzniklé nesprávným využitím šifry vedoucí k úniku bitů proudu klíče
4. Slabiny v implementaci standardu – chyby, kterých se dopustili výrobci při nesprávné implementaci standardu Mifare Classic. Tyto chyby například mohou umožňovat vyšší míru zranitelnosti či může docházet k úniku vyššího množství informace o klíči či čistém textu.

3.4.1 Nízká entropie generátoru pseudonáhodných čísel

Generátor pseudonáhodných čísel použitý v Mifare Classic má dvě místa zranitelnosti – krátkou generovanou sekvenci a resetování do výchozí hodnoty.

Popis zranitelnosti

Ke generování 32bitových hodnot nonce použitých pro autentizaci je použit PRNG, který je sice definován jako 32bitový LFSR, ale pro vytvoření následující hodnoty se využívá pouze část bitů a generátor se pak chová jako 16bitový (Garcia et al., 2008).

Generuje tak sekvenci 65536 čísel, která je podle (Nohl et al., 2008) vygenerována za 0.6 sekundy.

Dalším problémem je nesmyslné rozhodnutí návrhářů obvodu resetovat hodnotu do počáteční pevně dané hodnoty (Nohl et al., 2008). Toto rozhodnutí nejenže ničí náhodnost získanou předchozí komunikací a dělá tak generátor vlastně deterministickým, ale navíc vyžaduje navíc hardware pro nastavování počáteční hodnoty.

A poslední zneužitelnou vlastností tohoto návrhu je doba vytvoření nové hodnoty – u transpondéru s hodinovým signálem a u čtečky při požadavku na hodnotu.

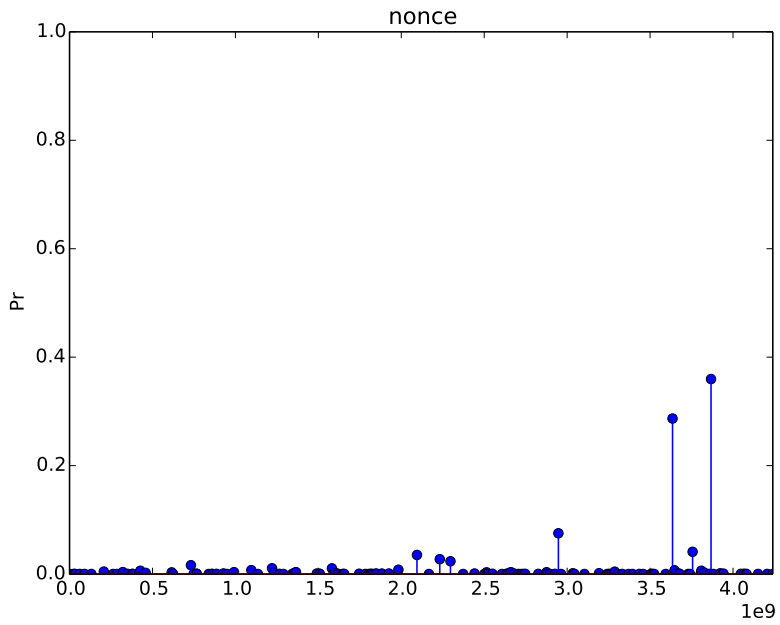
Bezpečnostní důsledek

Díky výše zmíněným nedostatkům v návrhu je možné provádět útoky na transpondér, poněvadž útočník je schopen pouhým vypínáním a zapínáním elektromagnetického pole kontrolovat hodnotu využívanou při autentizaci bloků.

Při útoku na čtečku je díky determinismu PRNG situace ještě jednodušší (za předpokladu, že není použit externí generátor pseudonáhodných čísel).

Možnost testování

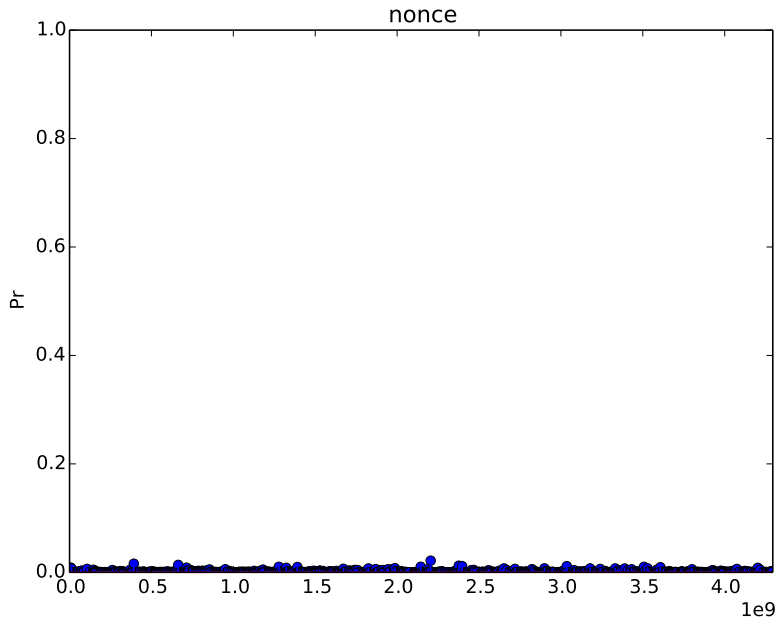
Při použití běžné čtečky namísto specializovaného hardwaru je obtížné dosáhnout přesného řízení časových intervalů mezi zapnutím a vypnutím generátoru.



Obrázek 3.4: Četnosti hodnot bez virtualizace

Tento problém lze částečně kompenzovat vytvořením statistiky a dosáhnout konstatní hodnoty zhruba ve 35 % případů. Hodnota s touto pravděpodobností byla nalezena vygenerováním 10000 náhodných hodnot a následným vypočtením histogramu (Obrázek 3.4).

Navíc byl objeven problém s časováním ve virtualizovaném systému pomocí nástroje VMWare Workstation 10.0. Na obrázku 3.5 jsou zobrazeny četnosti jednotlivých hodnot při použití z virtualizovaného systému.



Obrázek 3.5: Četnosti hodnot s virtualizací

Útočník pak pro získání vhodné konstatní hodnoty nonce musí provést následující kroky:

1. Zapnutí napájení
2. Provedení antikolizího protokolu
3. Odeslání požadavku autentizace
4. Vypnutí napájení
5. Uložení hodnoty

Tento postup provede při použití generické čtečky 1000krát a vybere hodnotu s nejvyšší četností, kterou následně může použít pro další části útoku. V případě

použití specializovaného hardwaru není nutné provádět dotazů tolik, poněvadž je možné například pomocí čítačů v FPGA[4] generovat intervaly nesrovnatelně přesnější než na počítači bez systému reálného času.

Obdobný postup byl použit ve vytvořeném nástroji (kód 3.1) jako součást detekce vadných implementací standardu Mifare Classic.

Zdrojový kód 3.1: Získání hodnoty s nejvyšší četností výskytu

```
for(i = 0; i < noncesCount; i++) {
  nfc_device_set_property_bool(pnd, NP_ACTIVATE_FIELD, true);

  // Select tag
  anticol(pnd);

  // Get nonce
  sizeRx = transmit_bytes(pnd, dataTx, dataRx, 4);
  print_hex(dataRx, sizeRx);

  // Store nonce
  memcpy(nonces+cursor, dataRx, sizeRx);
  memset(dataRx, 0, sizeRx);
  cursor += sizeRx;

  if(sizeRx > MAX_FRAME_LEN) {
    sizeRx = 0;
  }

  memset(dataRx, 0, sizeRx);

  nfc_device_set_property_bool(pnd, NP_ACTIVATE_FIELD, false);
}
```

Návrh řešení

Na obou stranách by bylo vhodné zvýšit rozsah generátoru zohledněním druhé části bitů v generující funkci, avšak tato úprava není realizovatelná kvůli funkci ověřující platnost výzvy.

[4]Field Programmable Gate Array

Aby mohl být generátor pseudonáhodných čísel považován za bezpěčný, tak by bylo nutné jej učinit nedeterministickým odstraněním resetování do známé konstantní hodnoty a řídit na obou stranách generátor pouze hodinovým signálem. Prakticky by pak byla hodnota z generátoru ukládána do nevolatilní paměti a při spuštění by na tuto hodnotu byl vždy generátor inicializován.

3.4.2 Kontrola integrity dat

Mifare Classic používá na transportní vrstvě standard ISO/IEC 14443. V normě (ISO 14443-3) je předepsáno, že za každým přeneseným datovým bajtem musí následovat jeden kontrolní bit realizovaný lichou paritou.

Při návrhu aplikační vrstvy však došlo k chybnému předpokladu, že kontrola integrity může být prováděna na aplikační vrstvě místo na transportní vrstvě, jak bylo zamýšleno v normě (ISO 14443-3).

Popis zranitelnosti

Při implementaci kontroly integrity došlo ke vzniku následujících nedostatků

1. Výpočet paritních bitů se provádí z otevřeného textu. Tento přístup je nevhodný, poněvadž dochází k úniku informace o čistém textu z paritních bitů.

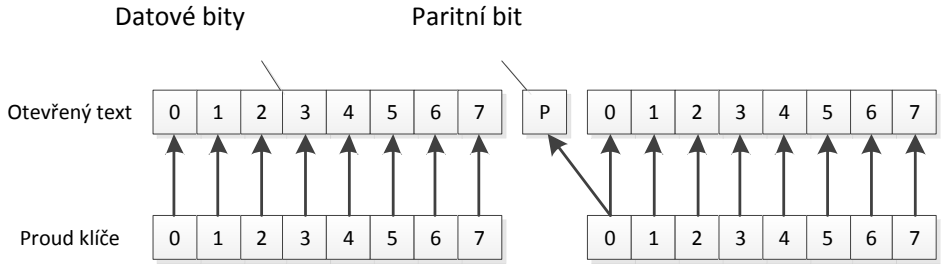
   Definice 7
   $p_j = n_{T,8j} \oplus ... \oplus n_{T,8j+7} \oplus 1$
   $p_{j+4} = n_{R,8j} \oplus ... \oplus n_{R,8j+7} \oplus 1$
   $p_{j+8} = a_{R,8j} \oplus ... \oplus a_{R,8j+7} \oplus 1$
   $p_{j+12} = a_{T,8j} \oplus ... \oplus a_{T,8j+7} \oplus 1$

2. Paritní bit je šifrován stejným bitem proudu klíče jako následující datový bit.

   Definice 8
   $\{p_j\} = p_j \oplus b_{8+8j}$

   Tento návrh šifrování porušuje jednorázovou tabulkovou šifru (one-time pad) tím, že využívá jeden stejný bit proudu klíče k šifrování dvou bitů čistého textu (Obrázek 3.6).

3. Kontrola integrity dat je prováděna ještě před ověřením znalosti sdíleného klíče protistranou. Je-li alespoň jeden paritní bit chybný, tak transpondér neodpoví a zablokuje se jako při provedení příkazu HALT.



Obrázek 3.6: Porušení one-time pad při šifrování paritního bitu

Podle Courtois (2009) při nastavení libovolné kombinace hodnot paritních bitů je pravděpodobnost 1/256, že transpondér odpoví zašifrovanou konstantní hodnotou NACK (0x5).

Jinými slovy při správném nastavení všech osmi paritních bitů a nesprávné odpovědi a_R transpondér odpovídá zašifrovanou konstantní hodnotou a útočník takto dokáže získat čtyři bity proudu klíče.

V práci (Courtois, 2009) byla publikována skutečnost, že některé karty jsou zranitelnější než jiné. Tyto slabší implementace totiž odpovídají hodnotou NACK s pravděpodobností 1 místo 1/256, neboli na každou kombinaci paritních bitů odpoví hodnotou NACK. Útočník takto získá čtyři bity proudu klíče s každým požadavkem a díky tomu je možné tyto karty prolomit během několika vteřin[5].

Bezpečnostní důsledek

Útočník může udržovat konstantní hodnotu výzvy n_T. V případě, že získaná hodnota n_T odpovídá zvolené hodnotě, tak zkusí následující možnost nastavení paritních bitů. S pravěpodobností 1/256 získá hodnotu $\{0x5\}$, ze které snadno získá čtyři po sobě jdoucí bity proudu klíče $\{0x5\} \oplus 0x5 = ks$. Pokud hodnota n_T neodpovídá zvolené hodnotě, tak transpondér resetuje.

Navíc v rámci autentizačního protokolu je nahrána výzva n_R do registru šifry protistrany. Když je takto útočník schopen odlišit výše zmíněné dva chybové stavy, tak je mu umožněno přímo manipulovat s generovaným proudem klíče. Tento postup byl využit například útokem publikovaným v (Garcia et al., 2009), kde je měněn pouze jeden bit výzvy n_R a sledováno, jestli je ovlivněn generovaný proud klíče. Tento

[5]Za předpokladu konstantní hodnoty n_T.

únik informace pak umožňuje najít určitou hodnotu výzvy, která zmenší prostor pro vyhledávání klíče hrubou silou.

Tento nedostatek je klíčový ve všech publikovaných útocích, například v (Courtois, 2009) je popsán útok využívající tuto slabinu ke sběru informací, které je dále možné využít k získání stavu registru šifry pomocí diferenciální kryptoanalýzy a následnému vypočtení použitého klíče.

Možnost testování

V nástroji byl implementován test pro detekci výše zmíněných slabých implementací. Nejdříve musí být zaručena platnost předpokladu konstantní hodnoty n_T (například řízením napájení) a následně jsou postupně odeslány dva požadavky s různými náhodnými kombinacemi paritních bitů.

Pokud na oba tyto požadavky odpoví transpondér hodnotou NACK, tak se jistě jedná o slabší implementaci, poněvadž pravděpodobnost je vyšší než 1/256.

Funkce provádějící detekci chybných implementací je uvedena ve výpisu 3.2.

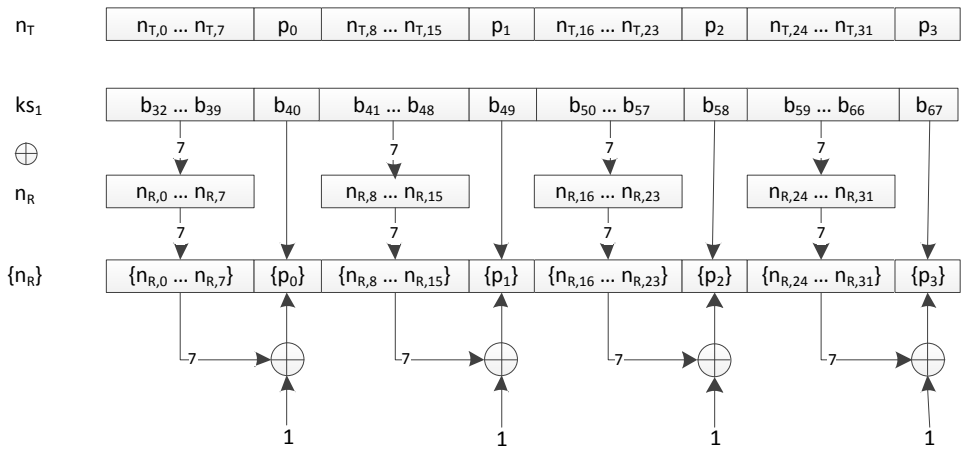
Zdrojový kód 3.2: Detekce slabých implementací

```
...
nfc_device_set_property_bool(pnd, NP_ACTIVATE_FIELD, true);

anticol(pnd);

// Get nonce
if( (sizeRx = transmit_bytes(pnd, dataTx, dataRx, 4)) < 0 ) {
  nfc_device_set_property_bool(pnd, NP_ACTIVATE_FIELD, false);
  run = false;
  continue;
}

//  When nonce is equal try parity
if(nt[0] == dataRx[0] && nt[1] == dataRx[1] && nt[2] == dataRx[2]
    && nt[3] == dataRx[3]) {
  // Try parity
  if( (sizeRx = transmit_bytes_as_bits(pnd, cryptogram, dataRx, 8,
      parity, NULL)) < 0 ) {
    nextVariation(parity, 8, 0);
```

```
18   } else {
19     okParity++;
20     run = (okParity > 1) ? false : true;
21   }
22 }
23
24 nfc_device_set_property_bool(pnd, NP_ACTIVATE_FIELD, false);
25 ...
```

Návrh řešení

Zranitelnost lze odstranit změnou šifrovacího schématu (3.7). V této změně je paritní bit počítán až z šifrového textu, aby nedocházelo k úniku informace, a dále je tento bit šifrován samostatným bitem proudu klíče (splňuje tedy jednorázovou tabulkovou šifru). Tato změna by ovšem také vyžadovala úpravy v řízení šifrovacího obvodu, poněvadž pro každý datový bajt je nutné vygenerovat jeden bit navíc pro šifrování parity.



Obrázek 3.7: Upravené šifrovací schéma

Teoreticky je odstranění této zranitelnosti jednoduché, ale prakticky ho realizovat lze jen s vysokými obtížemi, poněvadž změna protokolu je realizovatelná bez větších zásahů jen na straně čtečky[6] změnou softwaru v poskytované knihovně, SDK[7], nebo

[6]Nemusí nutně platit pro všechna zařízení
[7]Software Development Kit

přímo v konkrétní aplikaci. Na straně transpondéru by pak musel být vyměněn celý integrovaný obvod a navíc by dále bylo zřejmě nutné řešit problémy se zpětnou kompatibilitou.

Minimálně by však mělo být možné sjednotit odpověď při správné a nesprávné kombinaci paritních bitů, aby v obou případech došlo k zablokování, či odeslání jednotné chybové zprávy v čistém textu. Tuto vlastnost s nejvyšší pravděpodobností legitimní zařízení nevyužijí.

3.4.3 Výpočet proudu klíče pouze z lichých bitů

Aktuální hodnota proudu klíče je v šifře Crypto-1 závislá na aktuálním stavu posuvného registru. Zranitelnost spočívá v použití pouze lichých bitů stavu registru pro výpočet hodnoty proudu klíče pomocí výstupních filtrových funkcí.

Útočník takto může použít kryptoanalytickou metodu „Rozděl a panuj“ a prakticky tak zmenšit prohledávaný prostor možných stavů.

Popis zranitelnosti

Na obrázku 3.2 si lze povšimnout, že jsou pro výpočet hodnoty proudu klíče použity pouze liché bity 9, 11, 13, ..., 47.

V prvním kroku jsou všechny bity na svém místě a útočník může nalézt kombinaci 20 bitů pro liché bity.

Pro každou takto získanou kombinaci 20 bitů je vypočtena hodnota proudu klíče $(ks3)_0$ pomocí výstupní funkce. Jednotlivé kombinace mohou být vyloučeny pomocí porovnání hodnoty $(ks3)_0$ s hodnotou proudu klíče, která byla předem získána například využitím zranitelnosti v kontrole integrity (viz část 3.4.2).

Po provedení dvou rotací vlevo šifra využívá 19 bitů jako v předchozím případě. Při hledání kombinací lze výpočtem výstupní funkce získat $(ks3)_2$. Je tedy vidět, že je pro vytvoření těchto dvou bitů proudu klíče využito jen 21 bitů z registru šifry.

Pokud je stejný postup proveden i pro druhou sadu bitů s $(ks3)_1$ a $(ks3)_3$, tak lze výsledky snadno spojit, poněvadž nesdílejí žádné bity.

Nesprávné kombinace je možné vylučovat například díky znalosti několika po sobě jdoucích bitů proudu klíče získaných pomocí zranitelnosti při kontrole integrity.

Takto může být za určitých podmínek útočník schopen zrekonstruovat 39 bitů (resp. 40, když je počítán i bit ze zpětné vazby) stavu registru šifry a zbytek prohledat hrubou silou.

Bezpečnostní důsledek

Tato vlastnost umožňuje zkoušet klíče mnohem efektivněji než při prostém využití hrubé síly. Při prohledávání stavového prostoru hrubou silou existuje 2^{48} možných stavů, ale v (Courtois, 2009) byla využita tato vlastnost a bylo nutné vyzkoušet pouze 2^{16} možných hodnot (převinout tyto stavy zpět).

Dále byl v (Garcia et al., 2009) publikován útok, který pomocí této vlastnosti umožňuje nalézt speciální hodnotu výzvy n_R, jejíž poslední bit neovlivňuje proud klíče. Útočník při využití takovéto výzvy musí vyzkoušet jen $7,3 \cdot 10^9$ různých klíčů.

Možnost testování

Pomocí aplikace testovat tuto zranitelnost nemá smysl, poněvadž každé zařízení tohoto standardu bude zranitelné.

Návrh řešení

Řešením zranitelnosti by bylo odstranit toto rovnoměrné rozdělení na sudé a liché bity z pohledu generování hodnoty. Do výstupní funkce by měly být předávány bity ze sudých i lichých pozic. Navíc by bylo vhodné zajistit, aby bylo pro vytváření hodnot $ks0$ a $ks2$ ($ks1$ a $ks3$) využito co nejméně společných bitů.

Prakticky bohužel není možné tuto zranitelnost napravit, poněvadž by se pak jednalo o úplně jinou šifru. Bylo by pak nutné vyměnit všechna zařízení, či zavést režim kompatibility pro starší zařízení, která by ovšem zůstávala zranitelná.

3.4.4 Nejsou použité levé bity pro generování proudu klíče

Z obrázku 3.2 je patrné, že pro generování proudu klíče nejsou použity bity 0 až 8.

Popis zranitelnosti

Řekněme, že šifra má aktuální stav registru $r_k r_{k+1} ... r_{k+47}$ v určitém čase k. Útočník může použít vztah (Definice 9) k výpočtu předchozího stavu šifry $r_{k-1} r_k ... r_{k+46}$.

Pokud má útočník k dispozici stav šifry v čase k, hodnoty přenášené v otevřeném textu uid a n_T a hodnotu $\{n_R\}$. Poněvadž není známa hodnota n_R (známá je pouze její zašifrovaná hodnota), tak není možné provést „rollback“ šifry přímo.

Definice 9 *Rollback funkce* $R : F_2^{48} \to F_2$ *je definována jako*
$R(x_1, x_2, ..., x_{48}) = x_5 \oplus x_9 \oplus x_{10} \oplus x_{12} \oplus x_{14} \oplus x_{15} \oplus x_{17} \oplus x_{19} \oplus x_{24} \oplus x_{25} \oplus x_{27} \oplus x_{29} \oplus x_{35} \oplus x_{39} \oplus x_{41} \oplus x_{42} \oplus x_{43} \oplus x_{48}$

Při posuvu vpravo vypadne bit 47 a bit 0 je nastaven na hodnotu r. Poněvadž na prvních devíti bitech nezávisí proud klíče, tak na hodnotě r nezáleží a je možné ji volit náhodně. Následně je možné získat $n_{T,31} = \{n_{T,31}\} \oplus f_c(x_0, x_1, ..., x_{47})$ a vypočíst hodnotu $r_{i+80} = L(r_{i+32}, r_{i+33}, ..., r_{i+79}) \oplus n_{R,i}$. Pokud je tento postup zopakován ještě 31krát, tak se registr nachází ve stavu před načtením hodnoty výzvy čtečky n_R.

Následně lze obdobný postup využít dále pro $n_T \oplus uid$ a převinout šifru do výchozího stavu po přímém načtení šifrovacího klíče.

Tento postup byl publikován v práci (Garcia et al., 2008).

Bezpečnostní důsledek

Útočník je schopen postupně převinout stav šifry až do výchozího stavu, ve kterém je obsahem registru šifry pouze použitý klíč.

Možnost testování

Všechna zařízení budou zranitelná, proto nemá cenu testovat.

Návrh řešení

Přidat filtrovou funkci využívající bity na těchto pozicích. Je ovšem nutné aby výstup filtrové funkce měl rovnoměrné rozdělení (Tvrzení 1). Dále by musel být výstup zohledněn v samotné výstupní funkci f.

Poněvadž náprava spočívá přímo ve změně šifrovací funkce, tak opět vyvstává praktický problém se zachováním kompatibility se stávajícími zařízeními.

3.4.5 Malá variabilita proudu klíče

Jedná se o vlastnost použité logické funkce spočívající v tom, že výstupní bit funkce s určitou pravděpodobností nezávisí na více vstupních bitech.

Popis zranitelnosti

Mějme konstantní výzvu karty n_T a 8bajtový kryptogram $C = (c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8) = (n_R, a_R)$. První tři bajty $\{n_R\}$ budou fixně nastavené a u posledního bajtu se mohou měnit nejnižší tři bity.

Pro dotaz se správně nastavenou kombinací paritních bitů je při dešifrování pravděpodobnost 0,75, že proud klíče $ks_1 = (b_{32}, ..., b_{64})$ bude stejný bez ohledu na hodnotu $n_T$tj., nezávislý na posledních tři bitech n_T.

Bezpečnostní důsledek

Pokud vytvářený proud klíče je konstatní a tedy nezávisí na čtvrtém bajtu kryptogramu c_3, tak diference mezi stavem šifry při výpočtu ks_2 a ks_3 je nějaká lineární funkce závisející pouze na diferenci v hodnotách c_3 a ničem jiném.

Tato zranitelnost umožňuje provést útok pomocí diferenciální kryptoanalýzy popsaný v (Courtois, 2009).

Návrh řešení

Řešením by byla změna výstupní funkce tak, aby pravděpodobnost závislosti proudu klíče na vstupní hodnotě byla 0,5, avšak chyby samotné šifrovací funkce nelze opravit se zachováním kompatibility se stávajícími zařízeními. Jediným možným způsobem obrany je pak použití elektromagnetického stínění k zamezení komunikace karty s nelegitimními zařízeními.

3.4.6 Použití implicitních klíčů

Výrobce pro účely testování nastavuje při výrobě čipu implicitní klíče, se kterými jsou následně odeslány k prodeji.

Bezpečnostní důsledek

Pokud vývojář koncové aplikace tyto klíče nezmění, tak mohou být velice snadno přečteny, poněvadž implicitní klíče jsou veřejně dostupné – součástí knihovny LibNfc je tabulka nejčastějších implicitních klíčů.

Dále může být při úspěšné autentizaci jednoho bloku využit „Nested Attack“ publikovaný v práci (Garcia et al., 2009) k získání šifrovacích klíčů ostatních bloků.

Možnost testování

Pomocí slovníkového útoku lze snadno ověřit, zda je daný blok přístupný nějakým známým klíčem. Vytvořený nástroj využívá nejčastěji používané tovární klíče uvedené v tabulce 3.2.

Tabulka 3.2: Seznam implicitních klíčů

Hodnota klíče
0xffffffffffff
0xa0a1a2a3a4a5
0xb0b1b2b3b4b5
0x1a982c7e459a
0xd3f7d3f7d3f7
0xaabbccddeeff
0x000000000000
0x4d3a99c351dd

Návrh řešení

Řešením je prosté nastavení šifrovacích klíčů všem přítomným blokům. V případě, že má být karta využívána pro aplikace více subjekty, tak je nutné tyto klíče sdílet v rámci rozdělení paměťového prostoru jednotlivým aplikacím.

4 Návrh nástroje pro testování

4.1 Použité knihovny a zařízení

4.1.1 PN532 NFC/RFID controller breakout board

Jedná se o desku s čipem PN532. Tento čip je často využíván jako RFID řadič v mobilních telefonech a jiných embedded aplikacích.

Ke stolnímu počítači je možné desku připojit přes sériovou linku, případně k jiným systémům pomocí SPI[1] či I2C[2].

Při vývoji nebylo nutné kromě převodníku USB-UART používat další hardware, poněvadž deska vše potřebné obsahuje.

Deska s tímto čipem však nedokáže konkurovat speciálním zařízením jako je například Proxmark3, které dokáže díky integrovanému FPGA obvodu dodržet mnohem přesnější časové intervaly, výpočty urychlovat hardwarově, či odposlouchávat komunikaci mezi legitimní čtečkou a transpondérem. Tato deska je však řádově levnější a dokáže pro účely této práce plně nahradit jakoukoliv obyčejnou čtečku.

4.1.2 Libnfc 1.7.0

Knihovna Libnfc je nízkoúrovňová, multiplatformní a svobodná knihovna pro programování RFID zařízení.

Výhodou této knihovny je možnost pracovat s RFID zařízeními na té nejnižší úrovni přímým posíláním příkazů. Běžnou komunikaci jako je například čtení či zápis je možné nechat provádět přímo ovladač v jádře a čip PN532.

Složitější operace, jako je například udržování konstantní hodnoty n_T, autentizace, či samotné útoky je pak možné provádět přímým posíláním dat. Čip pak slouží pouze ke generování modulace a veškerá komunikace je řízena softwarově.

[1] Serial Peripheral Interface
[2] Inter-Integrated Circuit

Libnfc byla zkompilována a provozována na distribuci Xubuntu 13.10 operačního systému GNU/Linux, ale mělo by být možné ji zkompilovat i například ve Windows.

4.1.3 Crapto1

Crapto1 je implementace proudové šifry Crypto1 v jazyce C. Kromě běžného šifrování a dešifrování je možné ji také použít pro navrácení stavu šifry o krok zpět, či za určitých podmínek převinout stav zpět až k počátku.

Tato knihovna však neobsahuje žádnou formu interakce s uživatelem ani hardwarem, kterou je nutné dále doprogramovat například pomocí vybrané knihovny Libnfc.

Crapto1 poskytuje následující funkce:

- crypto1_create(key) – inicializace šifry s daným klíčem
- crypto1_get_lfsr – vrací současný stav registru šifry.
- crypto1_word – vrací 32 bitů proudu klíče a aktualizue hodnotu registru.
- prng_successor – implementuje suc funkci pro výpočet odpovědi při autentizačním protokolu

4.1.4 wxPython a matplotlib

Pro vykreslování grafického uživatelského rozhraní byla použita knihovna wxPython. Knihovna poskytuje všechny základní komponenty pro vytvoření rozhraní a obsluhu událostí od uživatele.

Navíc tato knihovna podporuje integraci knihovny matplotlib pro vykreslování grafů obdobným způsobem jakým disponuje nástroj Matlab.

4.2 Funkce

Navržený nástroj slouží k testování bezpečnosti RFID transpondérů a aplikací, které využívají Mifare Classic. Z tohoto důvodu nebyly implementovány konkrétní známé útoky, které ovšem lze s vytvořenými nástroji s určitým úsilím realizovat, ale pouze funkce pro detekci konkrétních zranitelností.

Takto lze velice rychle vyloučit implementace, které nelze s tímto vybavením napadnout, například pomocí „Dark side“ útoku. Útok na nezranitelnou implementaci

může být spuštěn hodiny a nikdy se sám nezastaví, pokud tedy není implementováno omezení doby běhu. Ale ani takto nelze bez další analýzy rozhodnout, proč útok selhal.

Nástroj umožňuje provádět následující operace:

1. Testování entropie použitého PRNG
2. Testování implementací odpovídajících na každou kombinaci paritních bitů 0x5
3. Testování použití implicitních klíčů k zabránění takzvanému Nested útoku
4. Vizualizace obsahu paměti s možností manipulace pomocí datového modelu

4.3 Nízkoúrovňové funkce a wrapper pro Python

Libnfc je knihovna určená pro programovací jazyk C, proto pro použití v jazyce Python bylo nutné datové typy a všechny funkce napsané v C obalit wrapperem. Tento wrapper je napsán v Pythonu pomocí tříd z knihovny ctypes.

V kódu 4.1 je uvedena třída, která obaluje strukturu nfc_device a poskytuje metody pro obsluhu zařízení v Pythonu. Obdobným způsobem jsou řešeny třídy pro ostatní nutné struktury a samotné funkce napsané v jazyce C.

Bohužel napsání wraperu pro knihovnu Libnfc v plném rozsahu by bylo časově náročné, a proto také v této práci byla vytvořena jen malá část, která byla nezbytná. Ostatní části pak byly napsány v jazyce C a do Pythonu byly předány pouze výsledky.

Celkem samotný wrapper obsahuje 24 tříd.

4.3.1 NfcContext

Struktura nfc_context musí být vytvořena a inicializována jako první před jakoukoliv jinou prací s knihovnou Nfclib, poněvadž obsahuje interní nastavení a ukazatele na další důležité části knihovny. V Pythonu byla vytvořena třída NfcContext, která obsahuje samotnou strukturu a s ní logicky spřažené funkce (inicializace a rušení kontextu).

4.3.2 NfcDevice

Další důležitou částí je struktura nfc_device sloužící k připojení k samotnému NFC[3] zařízení. Dále obsahuje informace nutné pro řízení zařízení – typ, ovladač a samotné nastavení funkce čipu. V nástroji je vytvořena třída NfcDevice, která reprezentuje tuto strukturu a jako metody má příslušné funkce (inicializace, rušení příkazu, uvedení do nečinného stavu a uzavření zařízení).

4.3.3 Nízkoúrovňové funkce

V části aplikace napsané v jazyce C byly naprogramovány následující funkce:

- transmit_bytes – Umožňuje přenos dat po jednotlivých bajtech s tím, že paritní bity jsou přidávány a kontrolovány hardwarově. Lze využít integrovaného obvodu pro zvýšení přesnosti přesným řízením časování.
- transmit_bytes_par – Umožňuje přenos dat po jednotlivých bajtech s manuálním nastavením paritních bitů. Tato funkce je využívána při testování na zranitelnosti.
- transmit_bites – Pro účely antikolizního protokolu musela být vytvořena funkce pro přenos dat po jednotlivých bitech. Ochrana přenosu je rovněž automatická.
- anticol – Tato funkce realizuje antikolizní protokol a vybírá dostupný transpondér.
- read – Běžné čtení dat z transpondéru.
- write – Běžný zápis bloku do transpondéru.

Nad těmito funkcemi byl v Pythonu naprogramován wrapper a samotné třídy pro práci s NFC zařízením.

4.3.4 Ostatní části

Dále byl vytvořen wrapper pro:

- Struktury jednotlivých typů transpondérů (různé typy ISO 14443, Felica a Jewel)

[3]Near Field Communication

- Struktury pro datovou komunikaci (modulace, rychlost, cílové zařízení)
- Struktury specifické pro Mifare Classic

Zdrojový kód 4.1: Wrapper pro nfc_device

```
class NFC_DEVICE(Structure):
    DEVICE_NAME_LENGTH = 256
    DEVICE_PORT_LENGTH = 64

    _fields_ = [
                ( 'context', POINTER( NFC_CONTEXT ) ),
                ( 'name', c_char * DEVICE_NAME_LENGTH ),
                ( 'connstring', c_char * NFC_BUFSIZE_CONNSTRING),
                ( 'bCrc', c_bool ),
                ( 'bPar', c_bool ),
                ( 'bEasyFraming', c_bool ),
                ( 'bAutoIso14443_4', c_bool ),
                ( 'btSupportByte', c_uint8 ),
                ( 'last_error', c_int )
                ]

class NfcDevice(object):
    LIBPATH = "/usr/lib/libnfc.so"

    @property
    def device(self):
        return self._device

    @device.setter
    def device(self, device):
        self._device = device

    def __init__(self, context, connstring=None, libpath=LIBPATH):
        try:
            self.lib = CDLL(libpath)
        except OSError as e:
            raise e
```

```
33          else:
34              self.lib.nfc_open.restype = ctypes.POINTER( NFC_DEVICE
                    )
35              self.device = self.lib.nfc_open( context.context,
                    connstring )
36
37              if self.device == None:
38                  raise NfcError()
39
40      def close(self):
41          self.lib.nfc_close(self.device)
42          self.device = None
43
44      def abortCommand(self):
45          code = self.lib.nfc_abort_command(self.device)
46
47          if code != NfcError.NFC_SUCCESS:
48              raise NfcError(code)
49
50      def idle(self):
51          code = self.lib.nfc_idle(self.device)
52
53          if code != NfcError.NFC_SUCCESS:
54              raise NfcError(code)
```

4.4 Třídy pro práci s NFC zařízením

Nad wraperem, který obaluje kód knihovny Libnfc a kód v jazyce C, jsou postaveny třídy pro samotnou manipulaci s transpondérem.

4.4.1 MifareBlockStream

MifareBlockStream je základní třída realizující přístup k NFC zařízení. V konstruktoru je provedeno potřebné nastavení pro komunikaci s transpondérem (modulace, přenosová rychlost, cílová struktura) a dále je otevřena sdílená knihovna obsahující kód pro samotnou práci se zařízením.

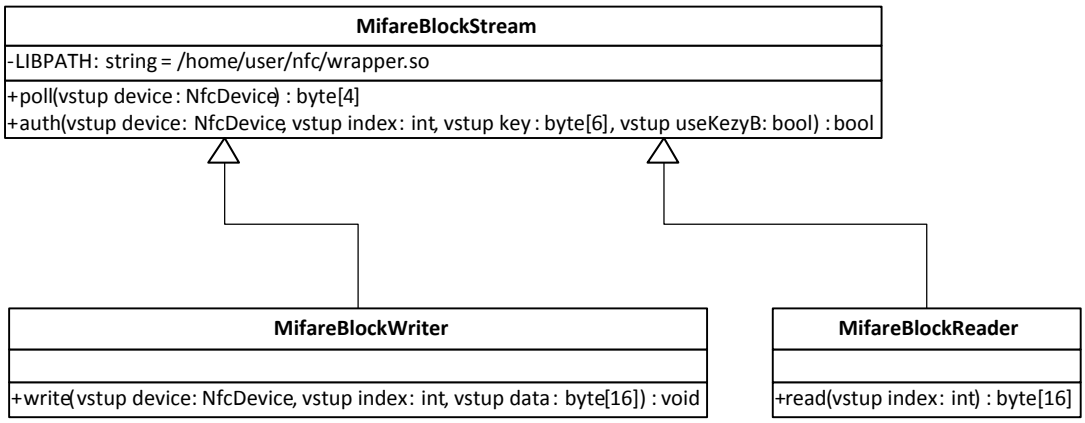
Pro práci s transpondérem poskytuje třída MifareBlockStream tyto metody:

- __init__(libPath) – inicializace knihovny a připravení datových struktur
- poll(device) – výběr transpondéru
- auth(device, index, key, useKeyB) – autentizace bloku

Zavoláním metody *poll(nfcDevice)* dojde k automatickému provedení antikolizního protokolu a tedy i výběru jednoho transpondéru. Při volání této metody je vykonána stejnojmenná funkce napsaná v jazyce C. Pokud nastane při vykonávání antikolizního protokolu chyba, tak je vyhozena výjimka.

Před jakoukoliv manipulací s daty transpondéru musí být konkrétní blok autentizován určitým klíčem. K tomu slouží metoda *auth(nfcDevice, index, key, useKeyB)*, která obaluje stejnojmennou funkci napsanou v C. Metoda umožňuje zvolit autentizační klíč pomocí boolovské proměnné useKeyB. V případě chyby je opět vyhozena výjimka MifareException s příslušnou chybovou zprávou.

Od této třídy dědí následující dvě třídy pro čtení a zápis dat RFID transpondéru. Hiearchie těchto tříd je zobrazena na obrázku 4.1.



Obrázek 4.1: Třídy pro manipulaci s pamětí transpondéru

4.4.2 MifareBlockReader

Třída MifareBlockReader slouží pro samotné čtení informací z paměti transpondéru, která za tímto účelem poskytuje metodu *read(device, index)*.

Parametr device je reference na instanci třídy NfcDevice a parametr index slouží ke specifikaci bloku, který má být přečten. Pokud jsou data úspěšně přečtena, tak metoda vrací list šestnácti jednobajtových hodnot. V případě chyby je vyhozena výjimka.

Tato metoda automaticky nespouští autentizační rutinu, takže je nutné ji před samotným čtením zavolat.

4.4.3 MifareBlockWriter

K zápisu dat do paměti transpondéru slouží třída MifareBlockwriter poskytující metodu *write(device, index, data)*. Parametr data je list jednobajtových hodnot o velikosti bloku – ve standardu Mifare Classic je to 16.

Opět je nutné nejdříve úspešně provést autentizaci bloku před zápisem. Není nutné manuálně testovat správnost formátu bloku, neboť tato metoda ji provádí automaticky. Nástroj tak nepovolí zápis poškozeného bloku do paměti karty, takže se není třeba obávat permanentního zablokování sektoru.

4.4.4 MifareException

Třída MifareException je ekvivalentem chybového kódu použitého částí aplikace napsané v jazyce C.

V případě, že nějaká funkce při přenosu dat končí chybou, tj. vrací zápornou hodnotu, tak je vytvořena nová instance třídy MifareException. Třída obsahuje metodu *errorMessage(code)*, která na základě obdrženého chybového kódu a tabulky 4.1 nastaví chybovou zprávu. Řízení je pak předáno do vyšší úrovně programu k dalšímu ošetření chybového stavu.

Tabulka 4.1: Chybové kódy

Chyba	Hodnota
NFC_SUCCESS	0
NFC_EIO	-1
NFC_EINVARG	-2
NFC_EDEVNOTSUPP	-3
NFC_ENOTSUCHDEV	-4
NFC_EOVFLOW	-5
NFC_ETIMEOUT	-6
NFC_EOPABORTED	-7
NFC_ENOTIMPL	-8
NFC_ETGRELEASED	-10
NFC_ERFTRANS	-20
NFC_EMFCAUTHFAIL	-30
NFC_ESOFT	-80
NFC_ECHIP	-90
DEFAULT_ERR_CODE	-69

4.5 Třídy pro testování zranitelnosti

4.5.1 BaseTest

BaseTest je abstraktní třída, která je předkem všech konkrétních tříd testů. Při dědění od této třídy všem potomkům předepsáno implementovat metodu *run()* sloužící k provedení testu.

Tato metoda vrací hodnotu, kterou lze vyhodnotit jako logická nepravda, pokud nebyl daný nedostatek detekován. V opačném případě vrací hodnotu vyhodnotitelnou jako logická pravda. Tento přístup umožňuje testům vracet různá data – typicky přímo logickou hodnotu, nebo pole bajtů.

4.5.2 PrngDetermTest

Hodnoty z generátoru pseudonáhodných čísel byly získány pomocí funkce *read_prngs(nfc_device *pnd, uint8_t **out, int cnt)*, která je součástí vytvořené knihovny v jazyce C (Kód 3.1). Tato funkce na dané místo v paměti postupně uloží získané čtyřbajtové hodnoty.

Při alokaci pole pomocí ctypes nastával problém se špatně uloženými daty – v určitých případech byla v paměti hodnota o něco nižší, než být měla. Tento problém byl vyřešen alokací potřebného paměťového prostoru už v části napsané v jazyku C.

V Pythonu bylo pak napsáno vyhodnocení náhodnosti pomocí histogramu. Program nejdříve získá pomocí funkce *read_prngs(nfc_device *pnd, uint8_t **out, int cnt)* pole čtyřbajtových hodnot a převede je do celočíselné podoby (list integerů). Následně je list seřazen a jsou vypočteny četnosti jednotlivých čísel.

4.5.3 DefaultKeysTest

Při testování bloku, zda je přístupný pomocí nějakého známého klíče, je používán standardní slovníkový útok.

Třída DefaultKeysTest má statický atribut keys, což je list nejběžnějších implicitních klíčů, se kterými výrobci testují funkčnost transpondéru. Tyto klíče jsou též dostupné například jako součást knihovny Libnfc.

Při testování je potřeba vytvořit instanci třídy MifareBlockReader a následně provést posloupnost následujících operací:

1. Výběr dalšího klíče
2. Zavolání metody *poll(device)*
3. Zavolání metody *auth(device, blockNo, key, keyB)*
4. Ukončení nebo návrat k bodu 1

Po provedení metody *poll(device)* je antikolizním protokolem vybrán transpondér. Následně je zavolána metoda *auth(device, blockNo, key, keyB)*, která se pokusí s daným klíčem autentizovat (provede tedy použitý autentizační protokol). Výstupem je logická hodnota, na jejímž základě je běh ukončen, nebo je tato sekvence opakována s dalším klíčem.

Antikolizní (uveden ve zkrácené verzi v kódu 4.2) a autentitační protokol je řešen na nižší úrovni – v knihovně naprogramované nad Libnfc v jazyce C.

Zdrojový kód 4.2: Funkce realizující antikolizní protokol

```
1  int anticol(nfc_device *pnd) {
2    uint8_t dataRx[MAX_FRAME_LEN] = {0};
3    uint8_t dataTx[MAX_FRAME_LEN] = {0};
4    size_t sizeRx = -1;
5
6    // Request type A
7    if( (sizeRx = transmit_bits(pnd, cmd_reqa, dataRx, 7) ) < 0 ) {
8      return STATUS_ERR;
9    }
10   memcpy(abtAtqa, dataRx, 2);
11
12   // Select all
13   if( (sizeRx = transmit_bytes(pnd, cmd_select_all, dataRx, 2) ) <
         0) {
14     return STATUS_ERR;
15   }
16
17   // Check answer
18   if ((dataRx[0]^dataRx[1]^dataRx[2]^dataRx[3]^dataRx[4]) != 0) {
```

```
19    return STATUS_ERR;
20  }
21  memcpy(tagUid, dataRx, sizeRx); // Save tag UID
22
23  //  Create Select UID request
24  memcpy(dataTx, cmd_select_uid, sizeof(cmd_select_uid)); // Add
        command
25  memcpy(dataTx + 2, dataRx, MIFARECL_UID_LEN);   // Add tag UID
26  iso14443a_crc_append(dataTx, 7);        //  Add CRC
27
28  //  Transmit select(uid) request
29  if( (sizeRx = transmit_bytes(pnd, dataTx, dataRx, 9) < 0) ) {
30    return STATUS_ERR;
31  }
32  abtSak = dataRx[0];
33  return STATUS_OK;
34 }
```

4.5.4 NackProbTest

Tato třída umožňuje testovat transpondér na chybně implementovanou kontrolu integrity. Ve skutečnosti je veškerý výkonný kód prováděn uvnitř funkce *mifare_check_nack_prob(nfc_device *pnd)* uvedené v kódu 3.2. Třída NackProbTest ji jen zpřístupňuje pro použití v Pythonu.

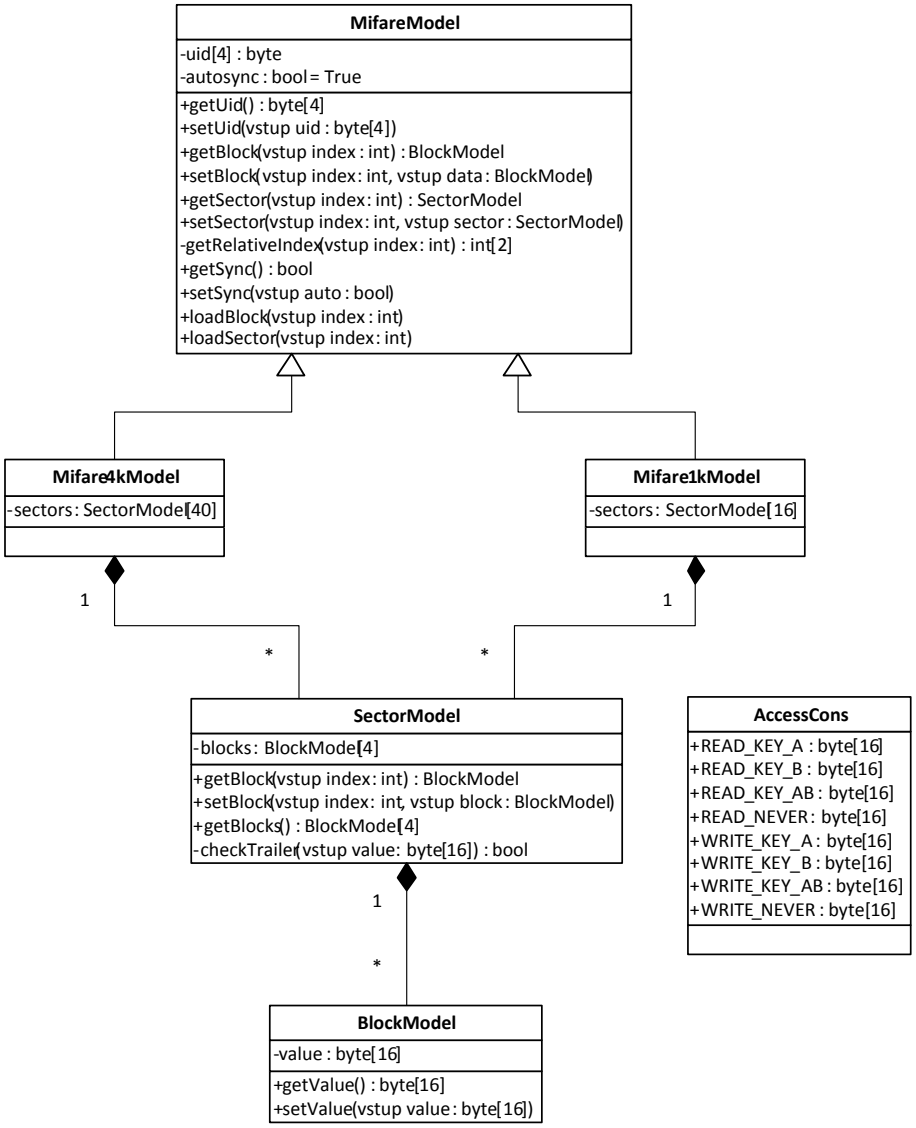
Při testování se využívá konstantní hodnota výzvy transpondéru n_T. Nejdříve je vybrán transpondér pomocí funkce *anticol(nfc_device *pnd)* a následně je započata první fáze autentizačního protokolu – získání hodnoty výzvy n_T z transpondéru. Pokud tato hodnota neodpovídá hodnotě konstatní výzvy, tak je karta zresetována vypnutím napájení.

V případě, že hodnota výzvy je konstatní, tak je vygenerován pomocí funkce *next_variation(uint8_t *data, size_t lenght, int fixed)* následující kombinace hodnot bitů. S těmito paritními bity je odeslán další požadavek s hodnotami $\{a_R\}\{n_R\}$. Pokud transpondér odpoví čtyřbitovou hodnotou, tak je zvýšen čítač správných kombinací paritních bitů o jedna.

Dále se postupuje stejně jako v předešlých odstavcích. Pokud je výsledná hodnota čítače vyšší než jedna, tak se jedná o zranitelnější implementaci standardu.

4.6 Datový model transpondéru

Na základě antikolizního protokolu získá čtečka informaci o typu transpondéru (SAK[4]). Při přijmutí hodnoty 0x08 se jedná o Mifare Classic s pamětí o velikosti 1 kB, a při přijmutí hodnoty 0x18 jde o 4 kB verzi transpondéru (NXP Semiconductors). Tuto skutečnost bylo nutné zohlednit i při návrhu programu.



Obrázek 4.2: Datový model

[4]Select Acknowledge, Type A

4.6.1 MifareModel

Ve chvíli, kdy se program pokusí poprvé načíst data z paměti transpondéru, tak musí být proveden jeho výběr antikolizním protokolem. Podle výsledku je vytvořena instance třídy Mifare1kModel, nebo instance třídy Mifare4kModel (Obrázek 4.2).

Dále je práce s těmito možnými transpondéry identická, liší se jen nutností vyhradit různě velké místo v paměti.

4.6.2 SectorModel

Každý Mifare Classic transpondér se skládá z určitého množství sektorů (záleží na typu), které jsou v programu reprezentovány třídou SectorModel. Poněvadž pro oba dva typy transpondérů je sektor stejného formátu, tak je možné mít pouze jednu třídu, která popíše oba typy. V této třídě je řešen přístup a manipulace s jednotlivými bloky s tím, že je použito přímé indexování bloků a přepočet indexu na sektor/blok probíhá interně.

SectorModel má implementovány metody pro přístup/manipulaci s bloky (getBlock, setBlock) a zodpovídá za vytvoření samotných bloků – instancí třídy BlockModel. Při zápisu bloku tato třída automaticky kontroluje formát, aby nemohlo dojít k nechtěnému permanentnímu zablokování sektoru z důvodu zápisu poškozeného konfiguračního bloku (metoda checkTrailer zobrazená v kódu 4.3).

Zdrojový kód 4.3: Kontrola formátu konfiguračního bloku

```
1 def checkTrailer(self, value):
2   testC1 = ( (value[7] & 0xF0) >> 4 ) ^ (value[6] & 0x0F)
3   testC2= ( (value[6] & 0xF0) >> 4 ) ^ (value[8] & 0x0F)
4   testC3 = ( (value[8] & 0xF0) >> 4 ) ^ (value[7] & 0x0F)
5
6   return (testC1 == 0x0f & testC2 == 0x0f & testC3 == 0x0f)
```

4.6.3 BlockModel

Třída BlockModel pak pouze obaluje 16bajtové pole a poskytuje metody pro manipulaci s daty.

Nastavení přístupových podmínek je obsaženo ve třídě AccessCons. Jedná se vlastně o kolekci dvojic podmínka a 16bajtové pole, které reprezentuje dané nastavení uložené v paměti transpondéru.

4.7 Prezentační vrstva

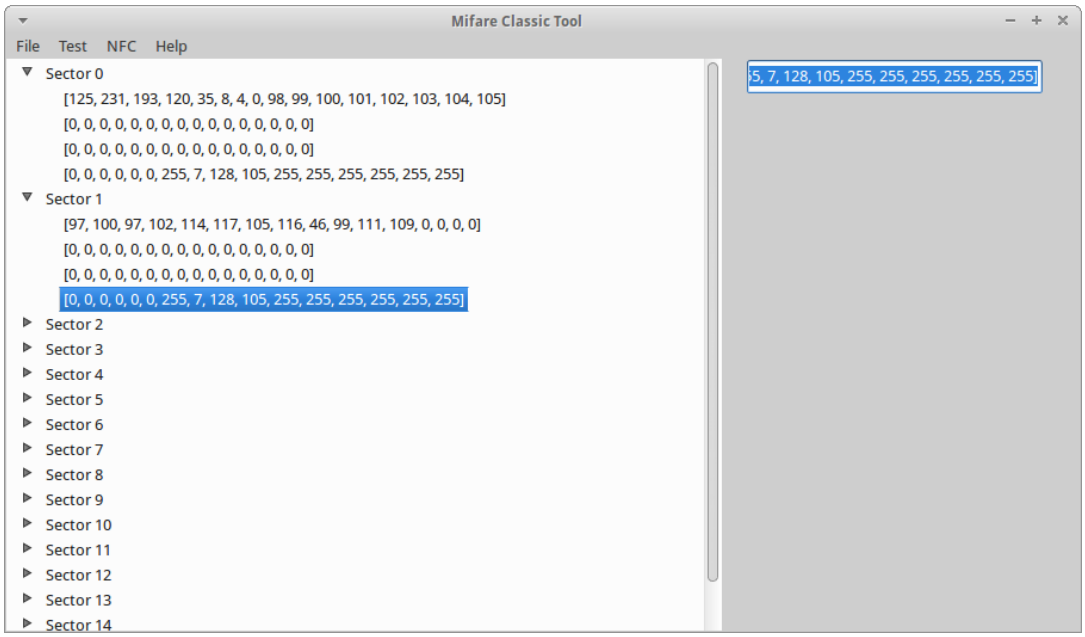
Grafické uživatelské rozhraní nástroje bylo vytvořeno za použití knihovny wxPython. Tato knihovna je postavena nad wxWidgets pro C++.

Aplikace se skládá ze čtyř formulářů a dialogů, které uživateli zpřístupňují dostupnou funkcionalitu.

4.7.1 Hlavní okno

Třída BasicFrame dědící od wx.Frame představuje hlavní okno aplikace. V levé části se nachází strom představující paměť transpondéru. V pravé části okna pak lze manipulovat s hodnotami uloženými v datovém modelu karty. Načítání hodnot a zápis hodnoty z editačního pole je možné provést z menu NFC.

Na obrázku 4.3 je zobrazeno hlavní okno aplikace. Na obrázku je aplikace po načtení hodnot prvních dvou sektorů. Paměť transpondéru je reprezentována komponentou wx.treectrl.



Obrázek 4.3: Hlavní okno s načtenými daty

Dále jsou z tohoto okna prostřednictvím roletového menu přístupné jednotlivé testy.

Poněvadž provádění samotných testů může trvat delší dobu, tak není vhodné testy spouštět v hlavním vláknu (blokování vykreslování grafického rozhraní). Z tohoto důvodu jsou všechny testy spouštěny v odděleném vláknu a výsledky jsou

předány do hlavního vlákna pomocí událostí (Kód 4.4). V události jsou obsaženy všechny nutné informace pro vyhodnocení zranitelnosti a prezentaci výsledku uživateli.

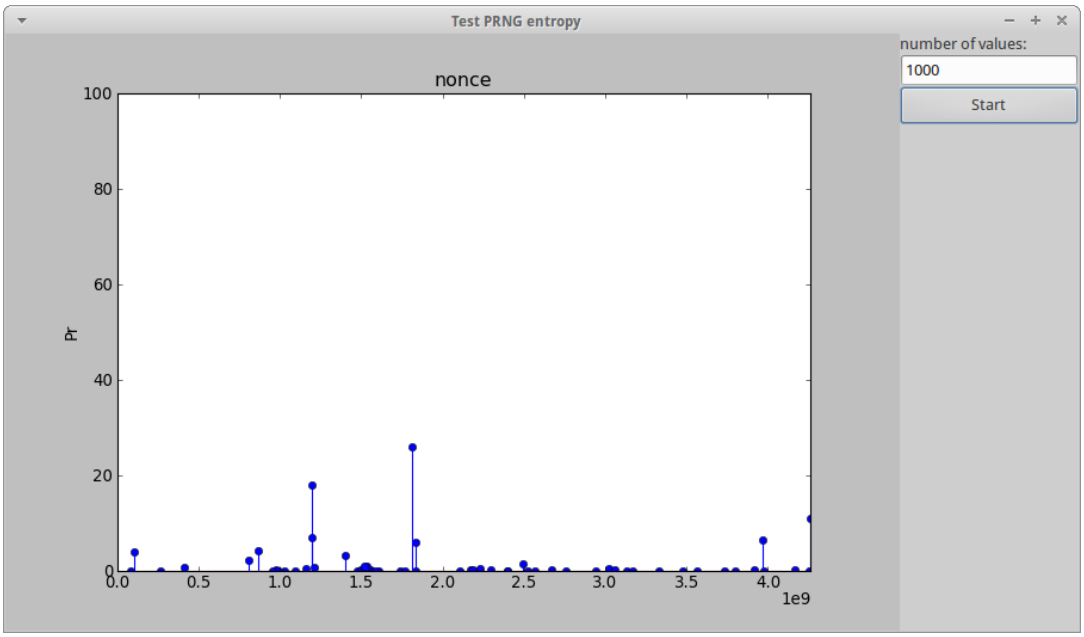
Zdrojový kód 4.4: Předávání hodnot z pracovních vláken pomocí událostí

```
def OnFinished(self, evt):
  '''
  Handles results from working threads
  '''
  message = None
  icon = None

  if evt._data:
    message = "Transponder is vulnerable!"
    icon = wx.ICON_WARNING

    if evt.GetValue() == self.EVT_TEST_KEYS:
      message = message + " Sector key = {}".format(evt._data)
  else:
    message = "Transponder is not vulnerable."
    icon = wx.ICON_INFORMATION

    # Show result in Alertbox
    wx.MessageBox(message, "Test {} is finished".format(evt._value
        ), wx.OK | icon)

def InitUI(self, parent):
  ...
  # Create event
  myEVT_WORK = wx.NewEventType()
  EVT_WORK = wx.PyEventBinder(myEVT_WORK, 1)

  # Bind event to OnFinished method
  self.Bind(EVT_WORK, self.OnFinished)
  ...
```

4.7.2 Okno pro testování entropie PRNG

Uživatel může provádět testy entropie PRNG z okna představovaného třídou PrngTestFrame.

V levé části tohoto okna se nachází graf vykreslený pomocí knihovny matplotlib. V pravé části okna se nachází textové pole, do kterého je zadán počet sbíraných hodnot z generátoru pseudonáhodných čísel (resp. počet odesílaných požadavků). Po dokončení běhu testu jsou získané hodnoty vykresleny do grafu (Obrázek 4.4).



Obrázek 4.4: Okno pro testování entropie PRNG

4.7.3 Dialogy

Aplikace využívá k získávání hodnot modální dialogy, do kterých uživatel vyplňuje potřebný údaj.

Třídy jednotlivých dialogů dědí od třídy wx.Dialog a jsou volány pomocí metody *dlg.ShowModal()*. Po odeslání dialogu je do hlavního okna vrácena návratová hodnota, která je wx.ID_OK v případě, že byl formulář odeslán. Další informací jsou samotná data, která jsou získána přímo z instance dialogu.

Třída DefaultKeyDialog slouží k získání čísla bloku, které má být použito pro spuštění testu z DefaultKeysTest.

K získání hodnoty výzvy a spuštění testu slabých implementací slouží dialog NackDialog.

5 Doporučení pro zabezpečení

Bohužel z důvodu kompatibility je prakticky téměř nemožné odstranit všechny nedostatky vedoucí ke zranitelnosti, poněvadž by bylo nutné vyměnit všechna zařízení nebo zavést mód kompatibility se staršími transpondéry a problém by tak přetrvával. Teoreticky byly však změny šifrovacího schématu popsány v části 3.4.

Existují však určité segmenty trhu, ve kterých je rozhodující především cena a bezpečnost je až na druhém místě. Pro tento případ zde byla sepsána metodika, která minimalizuje rizika spojená s nasazením takto slabě zabezpečené technologie.

Doporučení pro použití technologie Mifare Classic jsou:

1. **Použití pro nekritické aplikace**

   Nasazení ověřování uživatelů pomocí technologie Mifare Classic je nevhodné pro kritické aplikace – klíčové objekty jako jsou například elektrárny. Avšak při využití v méně kritických aplikacích, jako je například městská hromadná doprava, nemusí být vyšší míra zranitelnosti takovým problémem.

   V případě městské hromadné dopravy není finanční ztráta způsobená technicky vybaveným černým pasažérem ani zdaleka tak vysoká, jako jsou náklady spojené se změnou využívané technologie. Navíc je nutné počítat s vyšší cenou bezpečnější technologie.

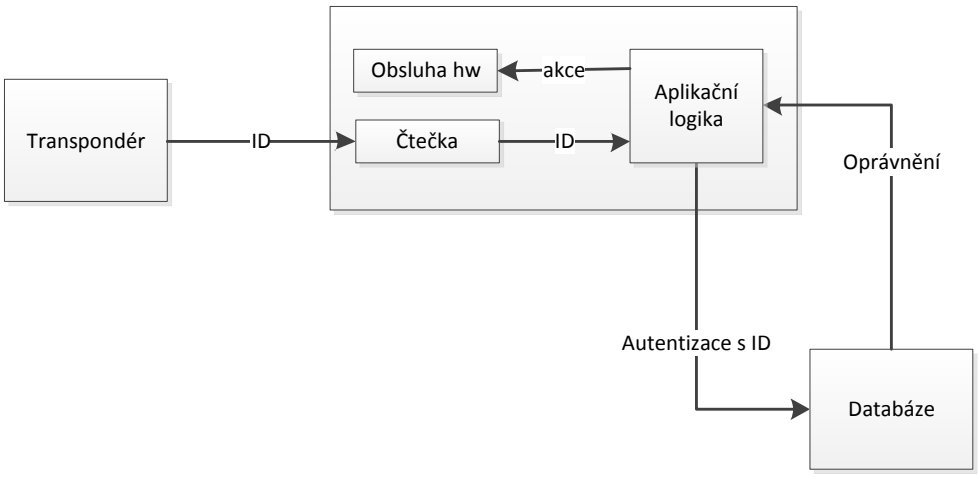
2. **Nastavení oprávnění**

   Minimalizovat riziko průniku do systému lze správným nastavením uživatelských oprávnění.

   Nejjednodušší a zároveň nejvíce náchylný přístup je nastavit do paměti karty jeden identifikátor (případně identifikátor oprávnění), kterým se uživatelé budou prokazovat. Bohužel pokud útočník získá tento identifikátor z jedné karty, tak dojde ke kompromitaci celého systému, či jeho části dané oprávněním– například bude mít neomezený přístup po celém objektu.

   Na kartu je pak vhodné uložit pouze identifikátor uživatele, který je předán čtečkou vyšší vrstvě. Tato vrstva se postará dále o autentizaci uživatele, například

oproti databázovému serveru, a případně dále přiřadí určitá oprávnění. Následně je provedena akce na základě přiřazených oprávnění a typu zařízení. Tento přístup je znázorněn na obrázku 5.1.



Obrázek 5.1: Autorizace uživatele pomocí RFID karty

3. **Filtrování vstupů**

   Je nutné uplatňovat stejné filtrování vstupů do databáze, jak z webové aplikace, tak z RFID čteček. Pokud útočník může manipulovat s pamětí karty, tak může být schopen přidat do její paměti nebezpečný kód.

4. **Omezení dosahu**

   Rádiové čtečky je vhodné nastavit tak, aby byl omezen jejich vysílací dosah a to nastavením nejmenšího možného vysílacího výkonu, případně vhodným umístěním antén. Takto lze útočníkovi ztížit možnosti odposlechu a případné další útoky.

5. **Uložení dat**

   Je nutné pečlivě zvážit kolik dat ukládat v paměti karty. Typicky by se mělo jednat o nejmenší možné množství informace, poněvadž v případě zranitelné technologie lze tyto informace považovat za dostupné útočníkovi.

   Další informace je pak vhodné v případě nutnosti načítat z databáze, nebo jiného vzdáleného a zabezpečeného uložiště.

6. **Stínění**

   Ve všech útocích na samotný transpondér musí mít útočník minimálně časově omezený přístup k transpondéru. Pokud není schopen nasbírat určité množství dat z komunikace s čipem karty, tak útok nemůže provést.

   Z tohoto důvodu je velice účinnou obranou prosté elektromagnetické stínění karty. S rozšířením pasů s RFID čipem není problém pořídit takto stíněné pouzdro na karty.

7. **Změna technologie**

   Přechod na bezpečnější technologii je logickým krokem k odstranění zranitelnosti.

   Na straně čtečky a aplikační logiky může být přechod poměrně bezbolestný, ponevadž existují čtečky operující na více standardech a ve stávající aplikaci je pak nutné pouze přidat obsluhu těchto požadavků.

   Postupně je pak možné vyměňovat transpondéry a migrovat kompletně směrem k bezpečnější technologii.

8. **Změna implicitních nastavení**

   Není vhodné spoléhat na implicitní nastavení od výrobce zařízení, protože tato konfigurace může být dostupná i útočníkovi (Thornton – Sanghera, 2011). Nejlepším řešením je změnit veškerou konfiguraci přesně na míru dané aplikaci.

6 Závěr

V této práci byly prozkoumány dosud publikované útoky na technologii Mifare Classic a to zejména ty cílené na samotný transpondér.

Po provedení analýzy byly popsány hlavní nedostatky v návrhu vedoucí ke zranitelnosti, bylo naznačeno možné zneužití a na základě získaných informací byla navržena metodika pro testování transpondérů.

V praktické části byl navržen a vytvořen v jazyce Python s použitím knihovny Libnfc nástroj pro testování transpondéru na dané nedostatky. Tato aplikace byla vyvíjena pro zařízení PN532 NFC/RFID controller breakout board, ale mělo by být možné jej použít s libovolnou čtečkou, která je kompatibilní s knihovnou Libnfc.

Z důvodu obtížného dodržování časových intervalů v běžném operačním systému by bylo vhodné namísto knihovny Libnfc zařadit mezi osobní počítač a čtečku ještě mikrokontrolér, který by prováděl časově kritické operace. V současném stavu je například možné dosáhnout konstatní hodnoty výzvy přinejlepším v 35 % případů a to může vést k delšímu běhu testu, poněvadž musí být odesláno mnohem více požadavků.

V rámci vývoje nástroje byly implementovány funkce pro běžnou komunikaci s transpondérem (výběr, autentizace, čtení a zápis) a funkce pro samotné testování zranitelnosti transpondéru (entropie PRNG, implicitní klíče, vadné implementace). Všechny tyto funkce je možné používat z grafického uživatelského rozhraní nástroje.

Byly navrženy metody pro odstranění zranitelnosti, ale lze je jen obtížně aplikovat kvůli obtížnému zachování zpětné kompatibility a nezanedbatelným finančním nákladům (platí zejména pro nedostatky v samotné šifrovací funkci).

Z tohoto důvodu byly navíc doporučeny obecné zásady pro použití této technologie tak, aby byla případná bezpečnostní rizika minimalizována.

Literatura

Courtois, N. T. *THE DARK SIDE OF SECURITY BY OBSCURITY and Cloning MiFare Classic Rail and Building Passes, Anywhere, Anytime*. Insticc-Inst Syst Technologies Information Control & Communication, 2009. WOS:000282088500052. ISBN 978-989-674-005-4.

Garcia, F. D. et al. Dismantling MIFARE Classic. In Jajodia, S. – Lopez, J. (Ed.) *Computer Security - Esoric 2008, Proceedings*, 5283. Springer-Verlag Berlin, 2008. s. 97–114. ISBN 978-3-540-88312-8, WOS:000262462900007.

Garcia, F. D. et al. Wirelessly Pickpocketing a Mifare Classic Card. In *Proceedings of the 2009 30th Ieee Symposium on Security and Privacy*. Los Alamitos: Ieee Computer Soc, 2009. s. 3–15. ISBN 978-0-7695-3633-0, WOS:000273336900001.

ISO 14443-3. *Identification cards – Contactless integrated circuit(s) cards – Proximity cards - Part 3: Initialization and anticollision*. Č. ISO 14443. ISO, Geneva, Switzerland, 2011.

Koning Gans, G. – Hoepman, J.-H. – Garcia, F. D. A Practical Attack on the MIFARE Classic. In *Proceedings of the 8th IFIP WG 8.8/11.2 International Conference on Smart Card Research and Advanced Applications*, CARDIS '08, s. 267–282, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-85892-8.

Nohl, K. Cryptanalysis of crypto-1. *Computer Science Department University of Virginia, White Paper*. 2008.

Nohl, K. et al. Reverse-engineering a Cryptographic RFID Tag. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, s. 185–193, Berkeley, CA, USA, 2008. USENIX Association.

NXP Semiconductors. *MF1S503x: MIFARE Classic 1K - Mainstream contactless smart card IC for fast and easy solution development*. NXP Semiconductors, 3.1 edition, February 2011.

NXP Semiconductors. *AN10833: MIFARE Type Identification Procedure.* NXP Semiconductors, 3.5 edition, March 2014.

NXP Semiconductors. *AN10927: MIFARE and handling of UIDs.* NXP Semiconductors, 3.1 edition, October .

Thornton, F. – Lanthem, C. *RFID Security.* Syngress, May 2006. ISBN 9780080489650.

Thornton, F. – Sanghera, P. *How to Cheat at Deploying and Securing RFID.* Syngress, April 2011. ISBN 9780080556895.

A Obsah přiloženého CD

Na přiloženém CD je uložen text práce ve formátu PDF. Dále jsou v adresářích obsaženy:

- dist – přeložená aktuální verze části aplikace napsáné v jazyce C
- src_c – zdrojové kódy části aplikace napsáné v jazyce C
- src_python – zdrojové kódy části aplikace napsáné v jazyce Python
- libs – použité knihovny, které nejsou dostupné z balíčkovacího systému (Libnfc 1.7.0, crapto)

B Postup přeložení Libnfc

Pro přeložení knihovny Libnfc je nutné provést následující kroky:

1. Rozbalení zdrojových kódů knihovny
   tar -xvzf libnfc-1.7.0.tar.gz
   cd libnfc-1.7.0

2. Konfigurace Libnfc pro PN532 a UART
   ./configure –with-drivers=pn532_uart –enable-serial-autoprobe

3. Překlad a instalace
   make clean
   make
   make install

Po provedení předchozích kroků lze funkčnost ověřit pomocí příkazu nfc-poll s oprávněními superuživatele (root).